WEB Development

45 Hours Training Program - Emerging Sector

Teaching - Learning Material

Project Implementation Unit

Department of Mechatronics, University of Engineering and Technology, Peshawar

Contents

Introduction

The Full-Stack Web Development training program is designed at the University of Engineering and Technology (UET), offering a demanding and fast-paced learning experience in modern web technologies and development practices. It equips trainees with a sound progressive learning path between basic and intermediate knowledge in modern Web Development, covering the MERN stack (MongoDB, Express.js, React.js, and Node.js). The program will allow the learners to develop scalable dynamic web applications that have frontend and backend development as the learners will be able to combine the rigorous instructions given with practical assignments. It is also keen on the industry relevant practices like the development of RESTful APIs, integration with databases, deployment, and portfolio creation, and, therefore, makes the graduates well equipped to take entry-level positions in the tech realm.

Training Objectives

1. A comprehensive understanding of the fundamental principles of Full-Stack Web Development, or, rather of such a stack as MERN includes the overall ownership of both frontend and backend programming: the former that consists of HTML, CSS, JavaScript, and React.js; the latter which includes Node.js, Express.js, and MongoDB.
2. Also significant are the creation of robust and safe RESTful APIs, the establishment of user authentication systems, as well as their handling of application information by means of uniform and pro digital backend provisions.
3. Also, developing reactive, dynamic web apps and running them on a cloud, like Heroku and Netlify, as well as developing a professional portfolio by doing practical and real-world projects, attests to such skills.

Training Learning Outcomes (TLOs)

TLO 1: The participants will be empowered to explain the main concepts of full-stack development, i.e. MERN stack architecture, including the frontend part (React.js), the backend systems (Node.js and Express.js), and the database (MongoDB). They will also be taught version control using Git and such deployment options as Netlify and Heroku.

TLO 2: Participants will also design, develop and deploy a fully constructed web application with the MERN stack. They will be able to understand how to integrate front and back end, how to create CRUD actions using mongoDB, how to do user authentication

Assessment

Component	Marks	Passing Criteria
Theory (MCQs + Short Questions)	30	50% (15 marks)
Practical (Capstone + Presentation)	70	60% (42 marks)
Total	100	To be eligible for the Certificate of Competency in Full-Stack Web Development, trainees must maintain at least 75% attendance and successfully pass both the theory and practical components of the assessment.

Should Enroll?

- **Beginners:** Individuals who initiate web development or continue their computer science, information technology, or similar studies in order to obtain practical experience in the area of full- stack development.

- **DAE Holders and Professionals:** Those who have a technical background and either seek further honing the skills or shift careers and make a transition to web reliant job roles using hard skills.
- **Technology Enthusiasts:** All those interested in creating and implementing modern web application with real project experience.

Training Module and Delivery plan

Total Training Hours	45 Hours
Training Methodology	**Theory**: 9 Hours (20%) **Practical**: 36 Hours (80%)
Medium of Instruction & Assessment	English & Urdu

Module 1: Health & Safety in Mobile App Development

1.1. Introduction

Web development is a major part of the modern technology industry, involving the design, coding, testing, and maintenance of websites and web applications. Although it may seem like a low-risk profession compared to fields such as construction or manufacturing, web developers also encounter unique health and safety challenges in their work environment.
Spending long hours at a computer can lead to eye strain, fatigue, and musculoskeletal issues such as neck, back, and wrist pain. Repetitive typing, poor posture, and insufficient breaks can cause injuries like carpal tunnel syndrome. In addition, stress from tight project deadlines, cybersecurity concerns, and electrical hazards from using multiple electronic devices can affect both physical and mental health.
Even small safety oversights, such as tangled power cords, overloaded sockets, or neglecting stress symptoms; can reduce productivity, disrupt teamwork, and cause long-term health problems. Therefore, maintaining a safe and healthy work environment is essential at every stage of web development, whether working in an office, a shared workspace, or remotely from home.
This module introduces key concepts of workplace safety, personal health practices, hazard awareness, emergency procedures, and basic first aid—all tailored to the needs of web developers in today's digital workspaces.

Module Objectives

By the end of this module, learners will be able to:

- Understand the importance of health and safety in web development environments.
- Apply ergonomic practices and maintain good posture while coding or designing.
- Recognize common workplace hazards in offices and remote setups.
- Follow safety and emergency procedures relevant to IT and digital workplaces.
- Demonstrate awareness of basic first aid for web developers and office professionals.

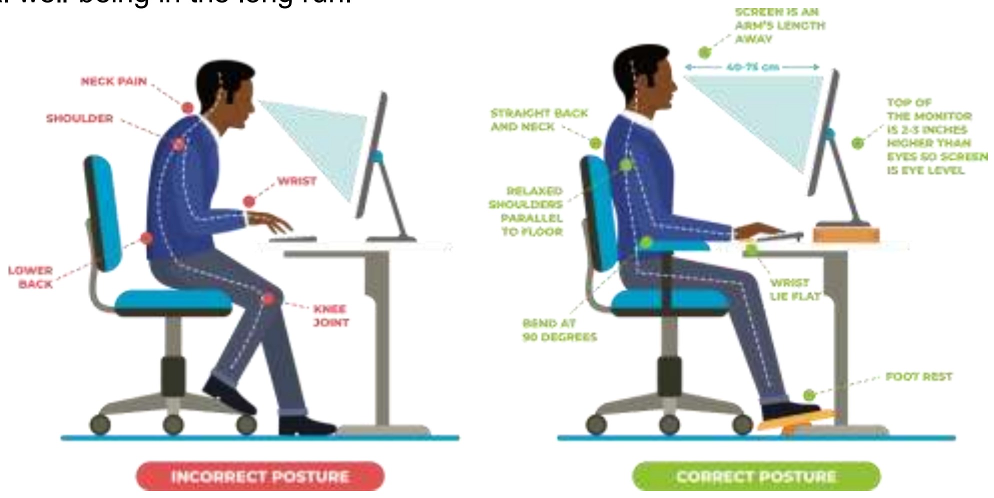
1.2. Learning Units (LUs)

LU1.2.1. Introduction to Safety in Labs

Web development environments come with specific safety and health challenges that require careful management. Developers often spend long hours sitting in front of screens, which can lead to ergonomic problems such as back pain, eye strain, and wrist fatigue. Electrical hazards from multiple connected devices, as well as cybersecurity risks from handling sensitive data, are also key concerns. A safe and efficient workspace should include an ergonomic setup—monitors positioned at eye level, adjustable chairs with proper back support, and sufficient lighting to reduce visual strain. Electrical safety measures, such as using surge protectors and keeping cables neatly organized, are equally important.

Cybersecurity safety in web development involves using secure coding practices, protecting login credentials, updating software regularly, and performing frequent data backups to prevent loss or breaches. Mental health is another vital aspect—developers should manage stress effectively, take regular breaks, and maintain a healthy work-life balance to avoid burnout.
By implementing safety protocols, emergency procedures, and routine equipment checks, web developers can create a secure, productive, and sustainable work environment that supports both personal well-being and project success.

LU1.2.2. Personal Safety Practices
Personal safety in web development involves maintaining both physical well-being and digital security. Developers should adopt proper ergonomic practices—sit with feet flat on the floor, keep the back supported, and maintain neutral wrist positions while typing or using a mouse. To reduce eye strain, follow the 20-20-20 rule: every 20 minutes, look at something 20 feet away for 20 seconds. Using blue light filters or glasses and ergonomic accessories such as adjustable chairs and keyboard supports can further enhance comfort.
Digital safety is equally important. Protect your accounts and data by using strong, unique passwords, enabling two-factor authentication, and performing regular data backups. Always follow secure coding practices, including input validation, secure data handling, and encryption, to safeguard against cyber threats.
In addition, take short breaks to stretch, move around, and manage stress throughout the workday. Ensure that your workspace has good ventilation and that equipment is kept clean and well-maintained.
By following these practices, web developers can prevent physical strain, protect sensitive information, and maintain consistent productivity—ensuring both high-quality code and personal well-being in the long run.



LU1.2.3. Hazards in the Workplace
Workplace hazards are conditions or situations that may cause harm, injury, or illness. In web development settings; whether in offices, co-working spaces, or home-based environments hazards can be both **visible** (like tangled cables or poor seating posture) and **hidden** (such

as eye strain, mental fatigue, or cybersecurity threats). Recognizing these risks is essential to maintaining a safe, healthy, and productive workspace.

Examples of Hazards in Web Development Environments:

- **Ergonomic Hazards:** Poor sitting posture, long hours without movement, or improper desk and chair setup leading to back pain, neck strain, or repetitive stress injuries.
- **Eye Strain & Screen Fatigue:** Continuous exposure to screens can cause blurred vision, headaches, and digital eye strain.
- **Slips, Trips & Falls:** Loose cables, cluttered work areas, or spilled beverages near electrical equipment.
- **Electrical Hazards:** Overloaded power sockets, overheating devices, and faulty adapters used for multiple monitors or servers.
- **Digital Hazards (Cybersecurity):** Malware from unsafe downloads, phishing emails, weak passwords, and insecure handling of website data or client information.
- **Noise & Distraction:** Open office noise, frequent notifications, or background interruptions that reduce concentration and increase stress.
- **Stress & Mental Fatigue:** Heavy workloads, tight project deadlines, or long coding sessions that lead to burnout or anxiety.
- **Housekeeping & Equipment Issues:** Cluttered desks, poor cable management, or dust accumulation inside computers causing overheating or reduced performance.
- **Health Risks from Poor Hygiene:** Shared equipment like keyboards, headsets, or devices that are not regularly sanitized.
- **Conflict & Aggression:** Miscommunication, work pressure, or interpersonal conflicts within development teams.

Key Rules for Web Developers:

- Follow all ergonomic, health, and digital safety guidelines provided by the organization.
- Immediately report unsafe conditions such as faulty equipment, health issues, or suspected cybersecurity threats.
- Consult supervisors or IT administrators when uncertain about safety or security procedures.
- Maintain both physical safety (clean, organized, and ergonomic workspace) and digital safety (secure coding, data protection, and proper device handling).

A strong safety culture in web development not only prevents injuries and data loss but also supports higher focus, creativity, and long-term career health.

LU1.2.4. Emergency Preparedness

Establish clear emergency protocols for fire, medical incidents, and data breaches. Maintain updated evacuation routes and assembly points. Implement automated backup systems with off-site storage. Create incident response plans for security breaches and system failures. Conduct regular emergency drills ensuring all team members understand procedures. Maintain emergency contact lists and first aid supplies. Document recovery procedures for data loss scenarios. Regular testing of backup systems ensures quick recovery from incidents. Preparedness minimizes downtime and protects both personnel and project assets during emergencies.

Focus on common development-related health issues. Eye strain relief includes the 20-20-20 rule and proper lighting. Repetitive strain prevention involves ergonomic setups and regular stretching. Stress management techniques include scheduled breaks and mindfulness practices. Maintain well-stocked first aid kits with digital eye strain relief. Train team members in basic emergency response. Address both physical and mental health concerns through proactive measures and immediate care protocols.

LU1.2.5. Basic First Aid Awareness

Emergency preparedness in web development involves planning and responding effectively to incidents that may threaten health, safety, or data integrity. Whether working in an office, a shared workspace, or remotely, developers must be ready to handle physical emergencies and digital crises alike.

Establish clear emergency protocols for situations such as fire, medical incidents, power outages, and data breaches. Maintain updated evacuation routes and clearly marked assembly points in office settings. Implement automated data backup systems with secure off-site or cloud storage to safeguard against data loss. Create and maintain incident response plans to address cybersecurity breaches, system failures, or ransomware attacks quickly and efficiently.

Conduct regular emergency drills to ensure all team members understand evacuation, communication, and response procedures. Keep an emergency contact list accessible and ensure that first aid kits are fully stocked and up to date. Document data recovery and restoration procedures and test backup systems periodically to verify reliability and minimize downtime after any incident.

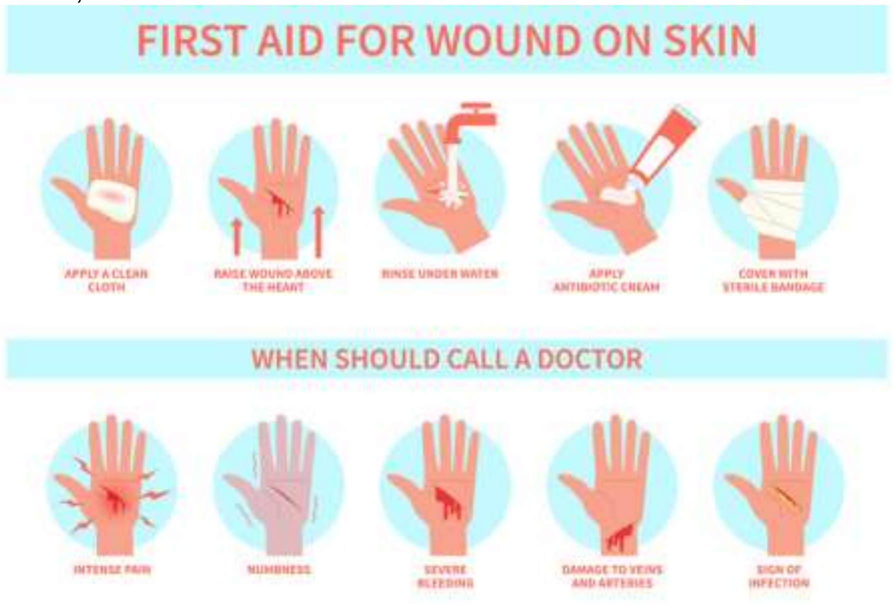
Key Topics

- **Introduction to CPR (Cardiopulmonary Resuscitation)**
  - CPR is a life-saving technique used when someone’s breathing or heartbeat has stopped.
  - Basic steps (for awareness only, not full certification):

CPR
SAVE A LIFE

    i. Check responsiveness – Tap and shout to see if the person responds.
    ii. Call for help – Dial emergency services immediately.
    iii. Check breathing – Look for chest movement.
    iv. Chest compressions – Place both hands in the center of the chest and press hard and fast (approx. 100–120 compressions pe r minute).
    v. Rescue breaths (if trained and comfortable) – After 30 compressions, give 2 rescue breaths.
    vi. Note: Only certified professionals should attempt CPR in real scenarios, but awareness of the process is vital.
- **Simple Care for Minor Injuries**
  - Cuts & Scrapes – Wash hands, clean wound with water, apply antiseptic, and cover with a bandage.

- Burns (mild from overheated devices or spilled hot liquids) – Cool burn under running water for at least 10 minutes, avoid applying creams or oils, cover with sterile gauze.
- Eye Irritation (dust, screen strain, accidental splash of cleaning solution) – Rinse eyes with clean water, rest, avoid rubbing eyes.
- Strains or Sprains – Use R.I.C.E. method: Rest, Ice, Compression, Elevation.
- Fainting/Stress Episodes – Lay the person down, elevate legs slightly, ensure fresh air circulation, and reassure them.


FIRST AID FOR WOUND ON SKIN
APPLY A CLEAN CLOTH
RAISE WOUND ABOVE THE HEART
RINSE UNDER WATER
APPLY ANTIBIOTIC CREAM
COVER WITH STERILE BANDAGE
WHEN SHOULD CALL A DOCTOR
INTENSE PAIN
NUMBNESS
SEVERE BLEEDING
DAMAGE TO VEINS AND ARTERIES
SIGN OF INFECTION

➢ **When and How to Seek Medical Help**

- Seek immediate professional help if:
  - The person is unconscious or unresponsive.
  - Bleeding does not stop after applying pressure for 10 minutes.
  - Burns cover a large area or are deep.
  - Chest pain, breathing difficulties, or signs of stroke appear.
  - Severe allergic reactions (swelling, difficulty breathing).
- Always know the emergency contact numbers (e.g., 1122 in Pakistan, 911 in US, 112 in EU).
- Report incidents to supervisor/lab manager immediately.

1.3. Practical Units (PUs)

PU 1.3.1. Setting Up an Ergonomic Web Development Workspace

Objective: To create a safe and comfortable workspace that prevents strain and supports long working hours.

Tasks:

- Arrange your workstation following ergonomic principles: monitor at eye level, proper chair height, and correct keyboard/mouse placement.
- Demonstrate correct sitting posture and hand positioning while coding.
- Identify poor ergonomic setups in sample workspace photos.
- Create a short checklist for maintaining daily ergonomic practices.

PU 1.3.2. Identifying and Reporting Workplace Hazards

Objective: To recognize common hazards in web development environments and develop awareness of safety protocols.

Tasks:

- Observe your computer lab or home workspace and list at least 10 possible physical, electrical, and digital hazards.
- Categorize each hazard (e.g., ergonomic, electrical, cybersecurity, environmental).

- Suggest preventive or corrective measures for each hazard identified.
- Prepare a brief report or presentation on maintaining a safe development workspace.

PU 1.3.3. Implementing Digital Security & Safe Coding Practices
Objective: To strengthen personal and organizational digital safety in web development.
Tasks:

- Create strong, unique passwords and enable two-factor authentication on development platforms (e.g., GitHub, email).
- Practice secure coding by applying input validation and data sanitization in a sample web form.
- Demonstrate how to perform regular data backups (local and cloud).
- Identify potential cybersecurity threats (e.g., phishing, malware) and describe response actions.

PU 1.3.4: Health & Wellness for Developers
Objective: To develop personal routines for physical and mental well-being while working on coding projects.
Tasks:

- Practice the 20-20-20 rule and demonstrate effective screen and lighting adjustments.
- Perform simple stretching or wrist exercises suitable for long working hours.
- Record stress management techniques (e.g., breathing, mindfulness, break schedules).
- Create a “Healthy Developer Routine” checklist combining physical and digital safety habits.

PU 1.3.5: Emergency Response & Data Recovery Drill
Objective: To apply emergency preparedness principles for both physical and digital incidents.
Tasks:

- Identify evacuation routes and emergency contact information for your workspace.
- Demonstrate how to respond to a mock fire or electrical emergency scenario.
- Create and test an automated backup and recovery plan for a small web project.
- Prepare a short-written emergency response plan for physical and digital incidents.

Module 2: Introduction to MERN Stack & Frontend Basics

2.1. Introduction

Modern web development relies on powerful technologies that allow developers to build dynamic, scalable, and interactive web applications. One of the most popular and efficient technology stacks used today is the MERN Stack, which includes MongoDB, Express.js, React.js, and Node.js. This full-stack JavaScript framework enables developers to use a single programming language; JavaScript, for both frontend and backend development, creating a seamless and efficient workflow.

The frontend of a web application focuses on what users see and interact with the design, layout, and interface elements. It involves using technologies such as HTML, CSS, and JavaScript to create responsive and user-friendly designs. In the MERN Stack, React.js plays a central role in building dynamic frontends that efficiently manage data and user interactions. On the other hand, the backend powered by Node.js and Express.js handles server logic, databases, and communication with the frontend. Together with MongoDB, a NoSQL database, these technologies form a complete environment for developing modern web applications.

This unit provides an overview of the MERN Stack architecture, introduces key frontend development principles, and helps learners understand how client-side and server-side technologies work together to deliver rich web experiences.

2.2. Learning Units (LUs)

LU2.2.1. Overview of Full-Stack Development and MERN Stack

In today's web-driven world, most digital applications rely on a combination of frontend and backend technologies to deliver smooth, interactive, and data-driven experiences. Full-stack development is the process of designing, developing, and managing both the client-side (frontend) and server-side (backend) components of a web application. A full-stack developer is a professional capable of handling the entire development cycle; from user interface design to database management and deployment.

This approach promotes better integration, faster problem-solving, and improved collaboration between teams since the developer understands the complete workflow of how data moves across the system.

2.2.1.1. Understanding Full-Stack Development

In a web application:

- The frontend is everything users see and interact with layouts, buttons, menus, forms, and overall user experience.
- The backend manages what happens behind the scenes; storing data, handling user authentication, processing requests, and communicating with the database.

A full-stack developer must understand how both layers interact through APIs (Application Programming Interfaces) and ensure that data flows securely and efficiently between the browser and the server.

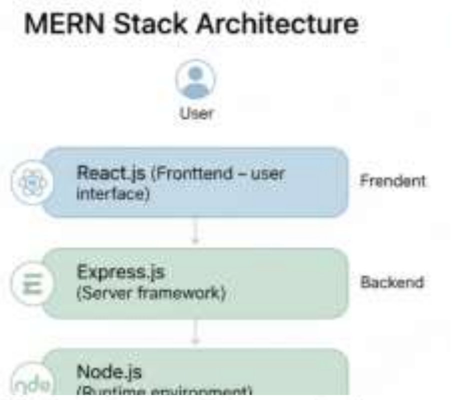
2.2.1.2. Key Skills of a Full-Stack Developer include:

- Frontend: HTML, CSS, JavaScript, and modern libraries like React.js.
- Backend: Server-side programming using Node.js and frameworks such as Express.js.
- Database Management: Handling data storage, retrieval, and updates through systems like MongoDB.
- Version Control & Deployment: Using Git, GitHub, and hosting platforms like Render, Vercel, or AWS.

2.2.1.3. Overview of the MERN Stack

The MERN Stack is one of the most widely used full-stack development frameworks. It is entirely based on JavaScript, making it easier for developers to use a single programming language across the entire project. The acronym MERN stands for:

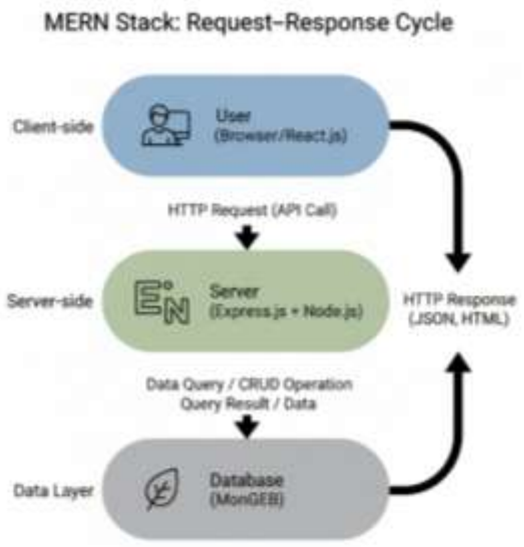
1. **MongoDB** – A NoSQL database that stores data in flexible, JSON-like documents rather than fixed tables. This allows for scalability and easy integration with JavaScript applications.
2. **Express.js** – A backend web framework for Node.js that simplifies handling routes, APIs, and server-side operations.
3. **React.js** – A frontend library developed by Facebook that enables developers to create dynamic, reusable UI components and efficiently manage application state.
4. **Node.js** – A JavaScript runtime environment that allows developers to execute JavaScript code on the server side for managing backend logic and communication with databases.


MERN Stack Architecture
User
React.js (Frontend – user interface)
Frendent
Express.js (Server framework)
Backend
Node.js (Runtime environment)

2.2.1.4. How MERN Stack Works Together

The MERN architecture follows a smooth data flow between its components:

1. **React.js (Frontend):** Manages the user interface, capturing user actions and sending requests to the backend.
2. **Express.js & Node.js (Backend):** Process these requests, apply logic, authenticate users, and interact with the database.
3. **MongoDB (Database):** Stores data such as user accounts, posts, orders, or product information and returns it when requested.


MERN Stack: Request–Response Cycle
Client-side
User (Browser/React.js)
HTTP Request (API Call)
Server-side
Server (Express.js + Node.js)
HTTP Response (JSON, HTML)
Data Query / CRUD Operation
Query Result / Data
Data Layer
Database (MonGEB)

This unified setup allows developers to create responsive, high-performance applications entirely using JavaScript.

2.2.1.5. Advantages of the MERN Stack

- **Single Language Efficiency:** Both frontend and backend use JavaScript, reducing complexity and learning curve.
- **Reusable Components:** React.js supports component-based architecture, improving development speed and consistency.
- **Scalability:** MongoDB's flexible schema allows easy scaling as the application grows.
- **Fast Development:** Node.js provides asynchronous processing for handling multiple user requests efficiently.
- **Open Source:** All technologies in the MERN stack are open source, supported by large developer communities.

2.2.1.6. Applications of MERN Stack

The MERN Stack is ideal for developing:

- **Social Media Applications** (e.g., community platforms, chat apps)
- **E-commerce Websites** (product catalogs, user authentication, and order management)
- **Project Management Dashboards**
- **Portfolio Websites and CMS Systems**
- **Real-Time Web Applications** (such as live chat and analytics dashboards)

LU2.2.2. Setting Up Development Environment (Node.js, VS Code)

Before beginning full-stack development using the MERN stack, it is essential to set up a proper development environment. A well-configured environment allows developers to efficiently write, test, and debug applications. Two key tools form the foundation of this setup; Node.js, which provides the backend runtime environment, and Visual Studio Code (VS Code), which serves as the primary code editor. This unit guides learners through installing and configuring these tools to create a smooth and productive workspace for MERN stack development.

2.2.2.1. Understanding the Development Environment

A development environment refers to the collection of tools, software, and configurations that allow developers to build and run applications effectively.

For MERN development, the environment typically includes:

- **Node.js** – for running backend JavaScript code.
- **npm (Node Package Manager)** – for installing external libraries and dependencies.
- **VS Code** – as the code editor.
- **Browser tools** (e.g., Chrome DevTools) – for debugging frontend code.
- **Git** – for version control and project management.

Proper setup ensures that developers can seamlessly switch between frontend and backend tasks using a unified workflow.

2.2.2.2. Installing Node.js

Node.js is a JavaScript runtime built on Chrome's V8 engine. It allows developers to execute JavaScript code outside the browser, making it essential for building server-side applications and APIs.

Steps to Install Node.js:

1. Visit the official website: https://nodejs.org
2. Download the LTS (Long-Term Support) version suitable for your operating system (Windows, macOS, or Linux).
3. Run the installer and follow the setup wizard.
4. During installation, ensure the option "Automatically install npm" is checked.
5. After installation, verify by opening the terminal or command prompt and typing:
6. node -v
7. npm -v

These commands display the installed versions of Node.js and npm, confirming successful installation.

2.2.2.3. Understanding npm (Node Package Manager)

npm is a built-in package manager with Node.js that allows developers to install, manage, and update external libraries or frameworks.

For example:

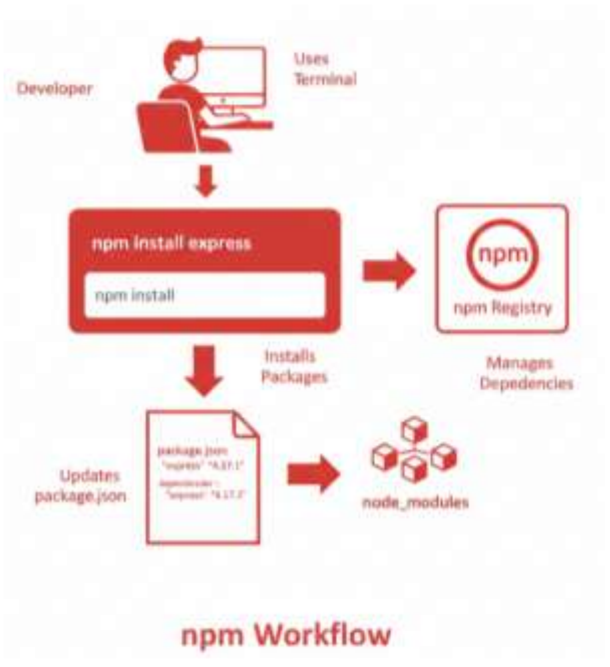
```
npm install express
```

This command installs Express.js into your project, adding it to the list of dependencies.

Common npm Commands:

- npm init – Initialize a new Node.js project.
- npm install <package> – Install a specific library.
- npm uninstall <package> – Remove an installed package.
- npm update – Update all project dependencies.
- npm run <script> – Execute predefined scripts in your project.

Understanding npm is crucial for managing dependencies and automating tasks like running the server or building the frontend.

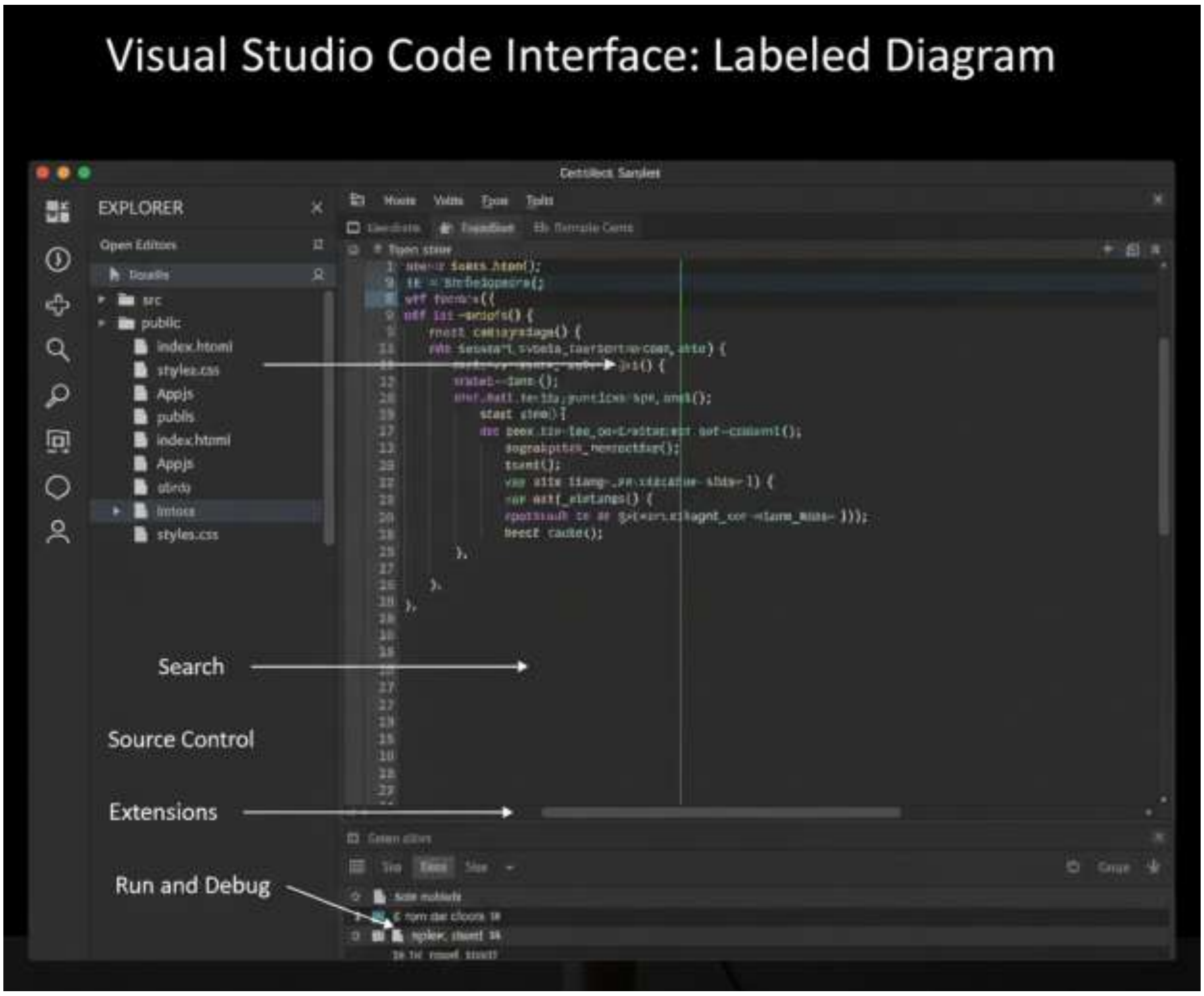


2.2.2.4. Setting Up Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is a lightweight yet powerful code editor widely used for web development. It supports JavaScript, Node.js, React.js, and integrates easily with Git for version control.

Steps to Install VS Code:

1. Visit https://code.visualstudio.com
2. Download the installer for your operating system.
3. Run the setup and complete installation.
4. Launch VS Code and explore its interface — including the Explorer, Search, Source Control, Run and Debug, and Extensions panels.



Recommended VS Code Extensions for MERN Development:

- **ES7+ React/Redux/React-Native snippets** – For React code shortcuts.
- **Prettier – Code Formatter** – For automatic and consistent code formatting.
- **Live Server** – To preview frontend files in real-time.
- **MongoDB for VS Code** – To connect and interact with MongoDB databases directly from the editor.
- **GitLens** – To enhance Git integration and visualize commit history.

These extensions streamline your workflow, improve code readability, and enhance productivity.

2.2.2.5. Creating a Simple Node.js Project

Once Node.js and VS Code are ready:

1. Create a new folder (e.g., my-first-app).
2. Open the folder in VS Code.
3. Initialize the project by typing:
4. npm init -y

This command creates a package.json file that stores project information and dependencies.

5. Create a file named app.js and add the following sample code:
6. console.log("Hello, Node.js Environment is ready!");
7. Run the application using:
8. node app.js

You should see this output:

Hello, Node.js Environment is ready!

This confirms that your Node.js environment is correctly installed and running.

LU2.2.3. Introduction to HTML5 and CSS Basics

2.2.3.1. Basics of HTML:

HTML5 (HyperText Markup Language, version 5) is the foundation of all modern web pages and applications. It provides the structure of a web page — defining how text, images, links, videos, and other elements are organized and displayed in a browser.

HTML5 introduced powerful features such as semantic elements, multimedia tags, and APIs that make web applications richer, faster, and more accessible.

Understanding HTML5 is the first step toward full-stack development, as it builds the foundation upon which CSS and JavaScript operate.

2.2.3.1.1. Purpose and Role of HTML5

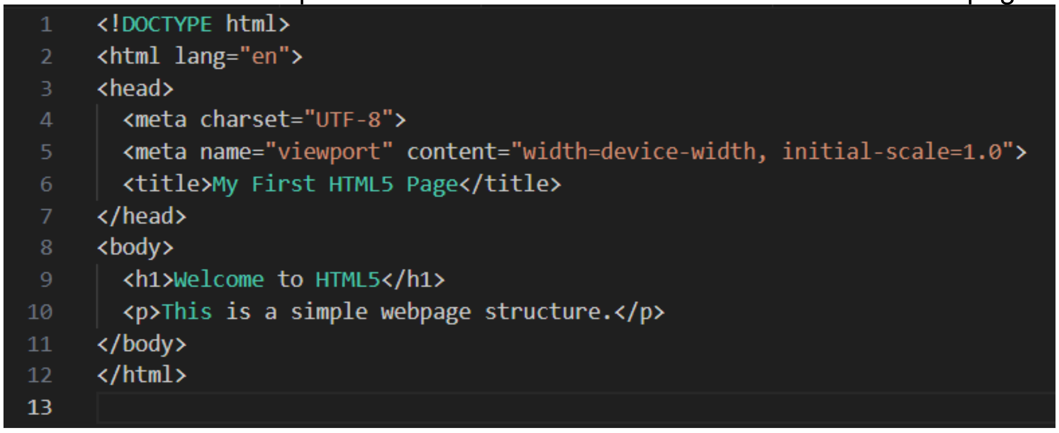
- Defines the structure and content of a webpage.
- Describes how browsers should display text, images, videos, and interactive forms.
- Provides semantic meaning to web content, improving accessibility and SEO.
- Serves as the base layer of every MERN stack application's frontend.

In a typical MERN app:

- HTML structures the layout.
- CSS styles it.
- React.js manipulates it dynamically via the Virtual DOM.

2.2.3.1.2. Basic Structure of an HTML5 Document

Every HTML5 file follows a specific structure. Here's the skeleton of a basic webpage:



```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My First HTML5 Page</title>
</head>
<body>
  <h1>Welcome to HTML5</h1>
  <p>This is a simple webpage structure.</p>
</body>
</html>
```

Explanation:

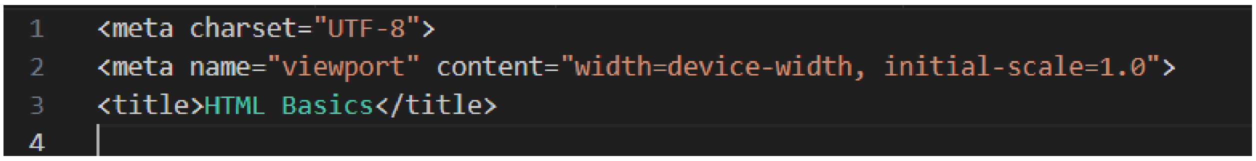
- <!DOCTYPE html> — Defines that the document is using HTML5.
- <html> — Root element of the page.
- <head> — Contains metadata, title, and linked resources (CSS/JS).
- <body> — Contains visible content (text, images, forms, etc.).

2.2.3.1.3. Core Components of HTML5

Metadata (Inside <head>**)**

Metadata describes information about the document, not displayed on the page.

Example:

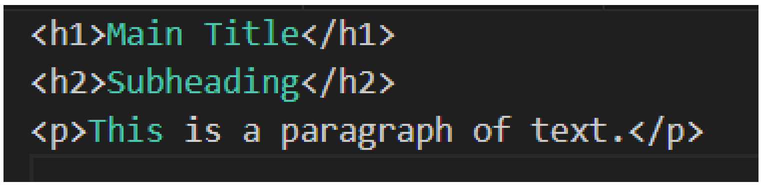


```html
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>HTML Basics</title>

```

- charset="UTF-8" → supports all characters.
- viewport → makes pages responsive on mobile.
- title → appears on the browser tab.

Text and Headings
HTML uses heading tags (<h1> to <h6>) for titles and <p> for paragraphs.

Example:



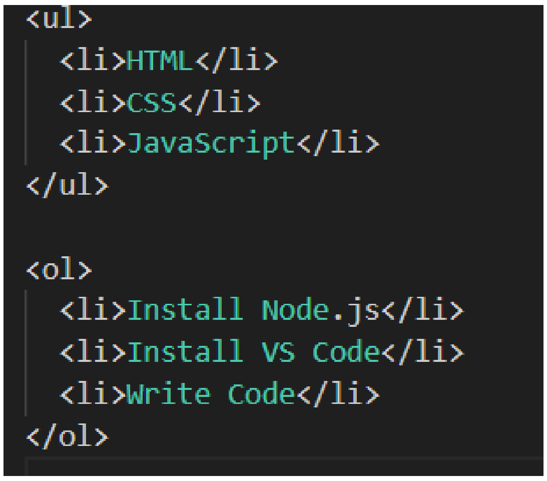
```html
<h1>Main Title</h1>
<h2>Subheading</h2>
<p>This is a paragraph of text.</p>
```

Links and Navigation
Links connect web pages using the <a> (anchor) tag.
Example: <a href="https://www.google.com" target="_blank">Go to Google</a>

- href — specifies the link address.
- target="_blank" — opens link in a new tab.

Lists (Ordered and Unordered)

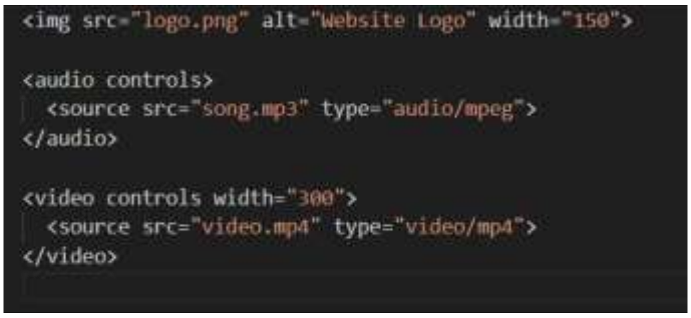
HTML supports numbered and bullet lists.



```html
<ul>
  <li>HTML</li>
  <li>CSS</li>
  <li>JavaScript</li>
</ul>

<ol>
  <li>Install Node.js</li>
  <li>Install VS Code</li>
  <li>Write Code</li>
</ol>
```

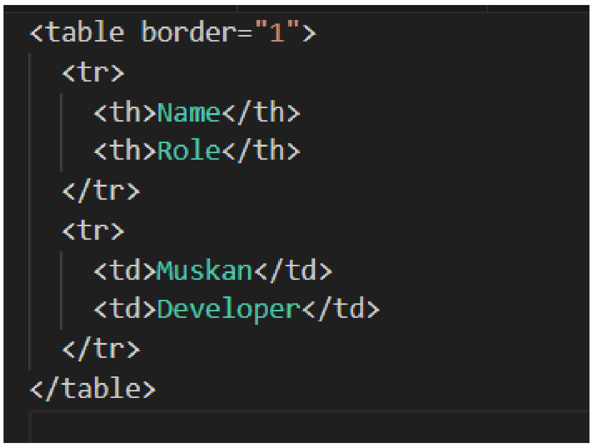
Images and Multimedia

HTML5 allows embedding of images, audio, and video directly.



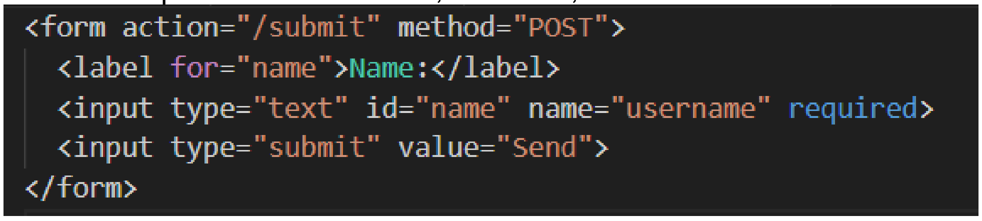
```html
<img src="logo.png" alt="Website Logo" width="150">

<audio controls>
  <source src="song.mp3" type="audio/mpeg">
</audio>

<video controls width="300">
  <source src="video.mp4" type="video/mp4">
</video>
```

Tables
Tables organize information in rows and columns.



```html
<table border="1">
  <tr>
    <th>Name</th>
    <th>Role</th>
  </tr>
  <tr>
    <td>Muskan</td>
    <td>Developer</td>
  </tr>
</table>
```

Forms and User Input
Forms collect user input for authentication, feedback, or data submission.

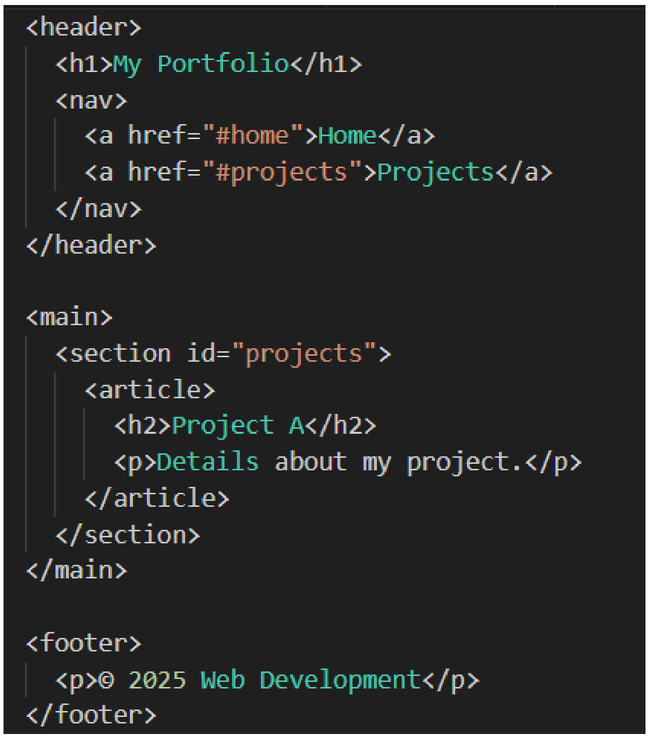


```html
<form action="/submit" method="POST">
  <label for="name">Name:</label>
  <input type="text" id="name" name="username" required>
  <input type="submit" value="Send">
</form>
```

Semantic Elements
Semantic elements describe the purpose of the content.
They make the page easier to understand by humans and machines.

Tag	Purpose
<header>	Page or section header
<nav>	Navigation links
<main>	Main content area
<section>	Logical group of related content
<article>	Independent content (blog, news post)
<aside>	Sidebar or supplementary content
<footer>	Page or section footer

Example:



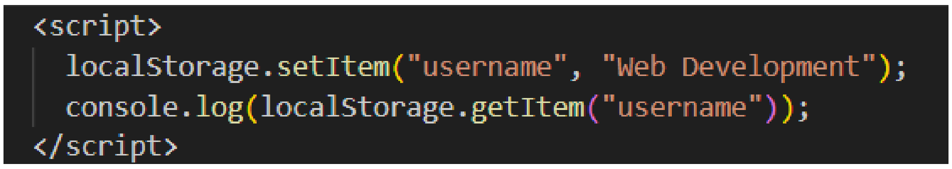
```
<header>
  <h1>My Portfolio</h1>
  <nav>
    <a href="#home">Home</a>
    <a href="#projects">Projects</a>
  </nav>
</header>

<main>
  <section id="projects">
    <article>
      <h2>Project A</h2>
      <p>Details about my project.</p>
    </article>
  </section>
</main>

<footer>
  <p>© 2025 Web Development</p>
</footer>
```

HTML5 APIs

HTML5 introduced built-in APIs for web applications:

- **Canvas API** → drawing graphics.
- **Geolocation API** → get user's location.
- **LocalStorage & SessionStorage** → store data locally.
- **Drag and Drop API** → enable drag functionality.

Example:



```
<script>
  localStorage.setItem("username", "Web Development");
  console.log(localStorage.getItem("username"));
</script>
```

Attributes in HTML5

Attributes provide additional information about elements.
They appear inside the start tag.

Example: <img src="photo.jpg" alt="A photo" width="200">

Attribute	Description
id	Unique identifier
class	For grouping and styling
style	Inline CSS styling
title	Tooltip text
src	Resource path
href	Link URL

Comments and Code Readability

Comments improve maintainability.

<!-- This is a comment explaining the section -->

Use comments to:

- Explain structure or logic.
- Divide sections of a long HTML document.

HTML5 Best Practices

✅ Always start with <!DOCTYPE html>
✅ Use semantic tags — avoid excessive <div> elements.
✅ Always close tags properly.
✅ Use lowercase for all tags and attributes.
✅ Include alt text for all images.
✅ Validate code using W3C Validator.
✅ Keep indentation consistent and readable.

Common Mistakes

❌ Missing closing tags.
❌ Forgetting the meta viewport for responsive design.
❌ Using outdated tags like <center> or <font>.
❌ Skipping alt text on images.
❌ Using <br> repeatedly for spacing instead of CSS.

2.2.3.2. CSS3 Basics

Cascading Style Sheets (CSS) is a stylesheet language used to describe how HTML elements are displayed on a web page. It controls the layout, colors, fonts, and overall appearance of a website. CSS separates the content (HTML) from design, allowing developers to maintain cleaner, more organized, and reusable code.
With CSS3, modern web design has evolved to include animations, gradients, shadows, flexible layouts, and responsive design; enhancing both aesthetics and usability.

2.2.3.2.1. Purpose of CSS

The main purpose of CSS is to define how HTML content should look in the browser.
It helps achieve:

- **Consistency:** Uniform design across all pages.
- **Efficiency:** Apply one stylesheet to multiple pages.
- **Flexibility:** Easily change design without editing every HTML file.
- **Responsiveness:** Adjust layout for mobile, tablet, and desktop screens.

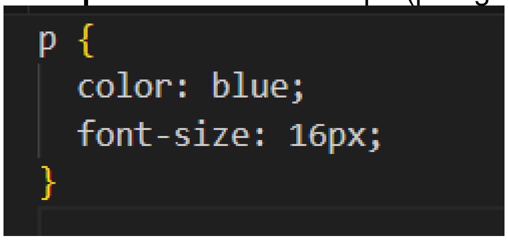
2.2.3.2.2. How CSS Works

CSS works by selecting HTML elements and applying styles to them.
The browser reads both HTML and CSS, then combines them to display the final styled webpage.
A CSS rule consists of:

```
selector {
  property: value;
}
```

- **Selector** identifies which HTML element to style (e.g., p, h1, .class, #id).
- **Property** defines what aspect to style (e.g., color, font-size).
- **Value** specifies how it should appear (e.g., blue, 16px).

Example: This makes all <p> (paragraph) text blue and 16 pixels in size.



```
p {
  color: blue;
  font-size: 16px;
}
```

2.2.3.2.3. Types of CSS

There are **three main ways** to add CSS to a webpage:

1. **Inline CSS** – written directly inside an HTML element using the style attribute.

```
<p style="color: green;">This is inline styled text.</p>
```

2. **Internal CSS** – written inside a <style> tag in the HTML <head> section.

```
<style>
  body {
    background-color: lightyellow;
  }
</style>
```

3. **External CSS** – stored in a separate .css file and linked using the <link> tag.

```
<link rel="stylesheet" href="styles.css">
```

External CSS is preferred for large projects because it keeps structure and design separate.

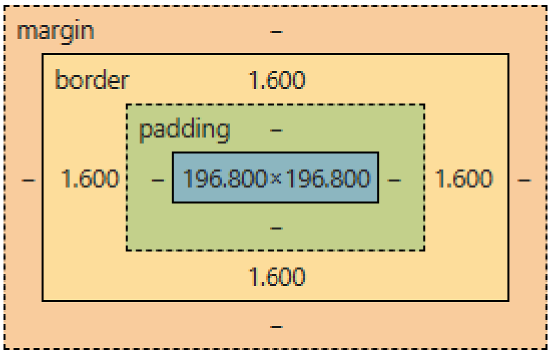
2.2.3.2.4. Common CSS Properties

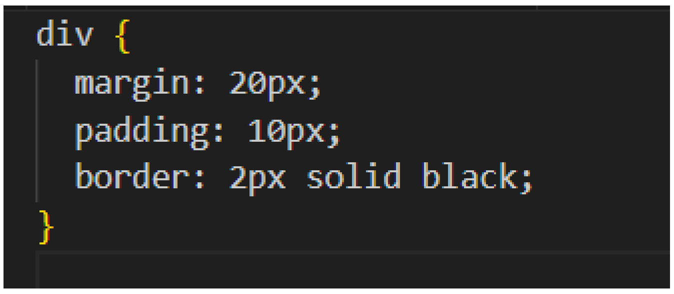
Category	Example Properties	Description
Text	color, font-family, font-size, text-align	Control typography and text appearance
Box Model	margin, padding, border, width, height	Define space and layout around elements
Background	background-color, background-image	Set colors or images behind elements
Positioning	position, top, left, float, display	Control how elements are arranged
Flexbox & Grid	display: flex;, display: grid;	Create modern, responsive layouts
Effects	box-shadow, border-radius, transition	Add visual enhancements

2.2.3.2.5. Understanding the CSS Box Model

Every HTML element is treated as a box consisting of:

1. **Content** – the text or image inside.
2. **Padding** – space between content and border.
3. **Border** – surrounds the padding.
4. **Margin** – space outside the border separating elements.





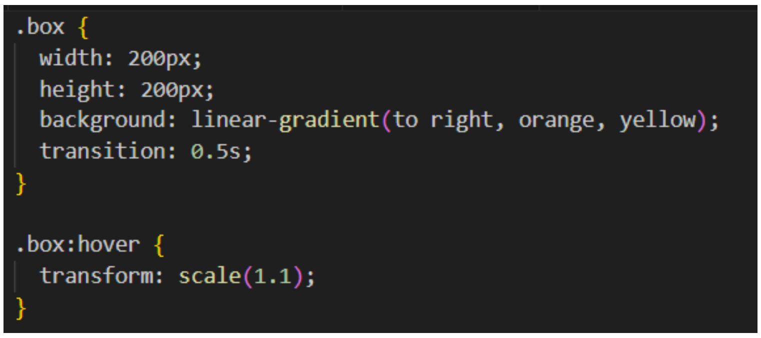
```
div {
  margin: 20px;
  padding: 10px;
  border: 2px solid black;
}
```

2.2.3.2.6. CSS3 Advanced Features

CSS3 introduced modern styling capabilities such as:

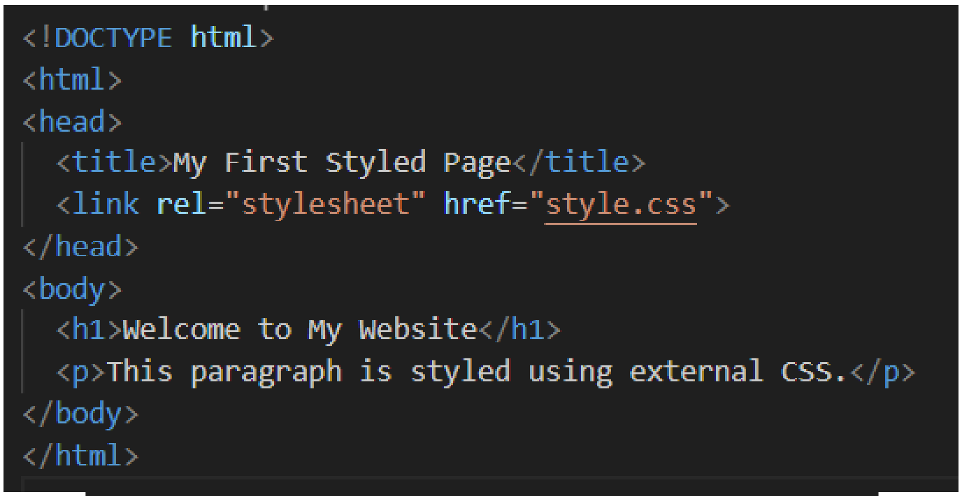
Feature	Example
Rounded Corners	border-radius: 10px;

Shadows	box-shadow: 2px 2px 10px gray;
Gradients	background: linear-gradient(to right, blue, green);
Transitions	Smooth animation effects
Media Queries	For responsive design
Flexbox & Grid	For flexible page layouts

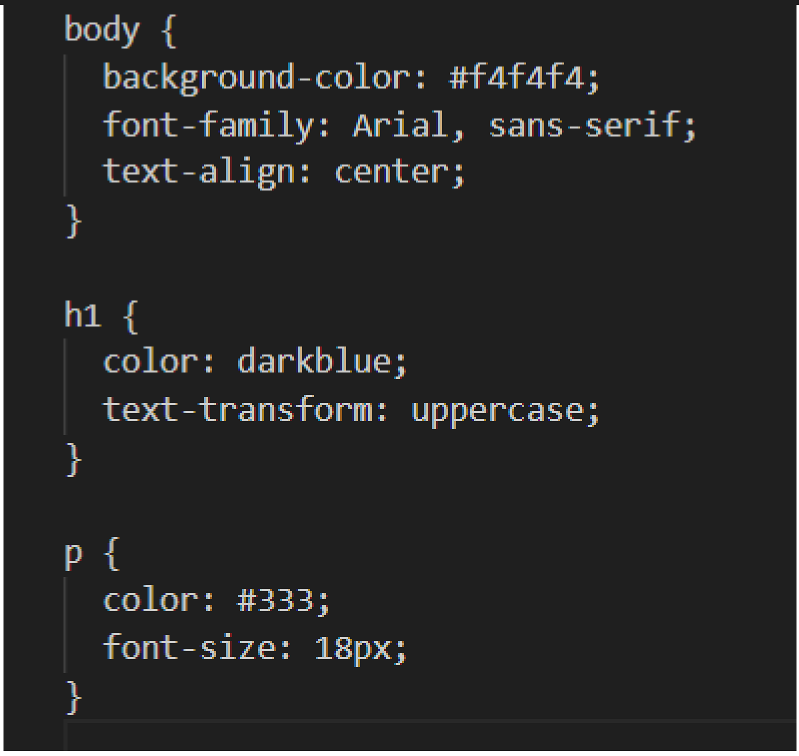


```css
.box {
  width: 200px;
  height: 200px;
  background: linear-gradient(to right, orange, yellow);
  transition: 0.5s;
}

.box:hover {
  transform: scale(1.1);
}
```

Example: HTML + CSS Combined

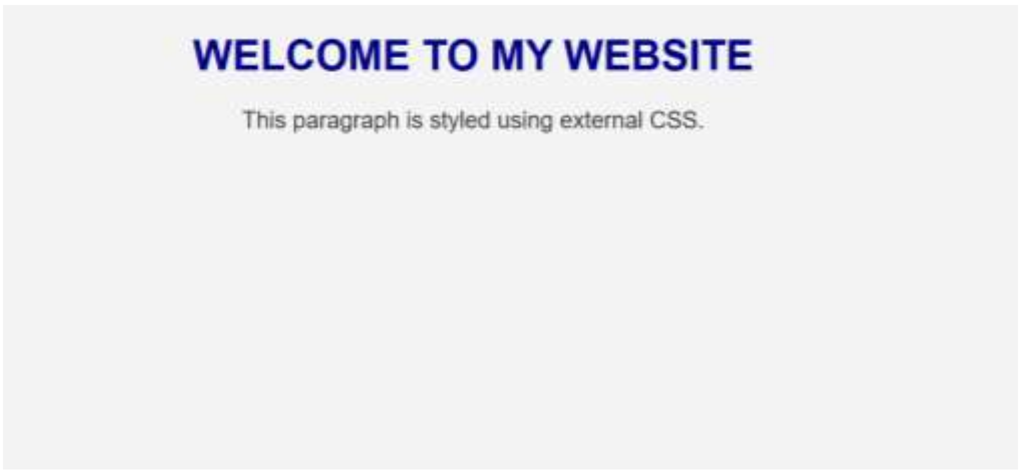


```html
<!DOCTYPE html>
<html>
<head>
  <title>My First Styled Page</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <h1>Welcome to My Website</h1>
  <p>This paragraph is styled using external CSS.</p>
</body>
</html>
```



```css
body {
  background-color: #f4f4f4;
  font-family: Arial, sans-serif;
  text-align: center;
}

h1 {
  color: darkblue;
  text-transform: uppercase;
}

p {
  color: #333;
  font-size: 18px;
}
```

Output:


WELCOME TO MY WEBSITE

This paragraph is styled using external CSS.

LU2.2.4. Building a Simple Web Page Layout

After understanding the basics of HTML5 and CSS3, the next step is to learn how to structure and design a simple web page layout. A web page layout defines how elements like the header, navigation bar, content area, sidebar, and footer are arranged visually on the screen. Creating a clear, well-organized layout improves both usability and user experience (UX), making it easier for visitors to navigate and interact with your website.

2.2.4.1. Basic Structure of a Web Page

Every standard webpage follows a general structure built using HTML5 semantic elements:

```
<!DOCTYPE html>
<html>
  <head>
    <title>My First Web Page Layout</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <header>
      <h1>My Website</h1>
    </header>

    <nav>
      <ul>
        <li><a href="#">Home</a></li>
        <li><a href="#">About</a></li>
        <li><a href="#">Contact</a></li>
      </ul>
    </nav>
    <main>
      <section>
        <h2>Welcome</h2>
        <p>This is a simple web page layout using HTML5 and CSS3.</p>
      </section>

      <aside>
        <h3>Sidebar</h3>
        <p>Additional information or links can go here.</p>
      </aside>
    </main>
    <footer>
      <p>&copy; 2025 My Website | All Rights Reserved</p>
    </footer>
  </body>
</html>
```

Explanation of Sections:

- **<header>** → Top section containing logo or website title.
- **<nav>** → Navigation bar with menus or links.
- **<main>** → Central area containing the main content of the page.
- **<section>** → Logical grouping of related content.
- **<aside>** → Sidebar for secondary content, ads, or links.
- **<footer>** → Bottom section containing contact info or copyright.

2.2.4.2. Styling the Layout with CSS

Now, let's add some CSS to make the layout look organized and professional.

style.css

```
body {
  font-family: Arial, sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f4f4f4;
}

/* Header */
header {
  background-color: #333;
  color: white;
  padding: 15px;
  text-align: center;
}
/* Navigation Bar */
nav {
  background-color: #444;
}

nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
  text-align: center;
}

nav ul li {
  display: inline-block;
  margin: 0 15px;
}

nav ul li a {
  color: white;
  text-decoration: none;
  font-weight: bold;
}

/* Main Content and Sidebar */
main {
  display: flex;
  margin: 20px;
}
```

```
section {
  flex: 3;
  background: white;
  padding: 20px;
  margin-right: 20px;
  border-radius: 5px;
}
aside {
  flex: 1;
  background: #e2e2e2;
  padding: 20px;
  border-radius: 5px;
}

/* Footer */
footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 10px 0;
}
```

Explanation:

- The layout uses Flexbox (display: flex) to align the main content and sidebar side-by-side.
- Navigation links are arranged horizontally using display: inline-block.
- Colors, margins, and padding improve the layout's visual hierarchy and readability.

2.2.4.3. Responsive Design with Media Queries

To make the layout mobile-friendly, you can use a simple media query:

```
@media screen and (max-width: 768px) {
  main {
    flex-direction: column;
  }
  section, aside {
    margin-right: 0;
    margin-bottom: 20px;
  }
}
```

This ensures that on smaller screens, the sidebar appears below the main content instead of beside it.

2.2.4.4. Folder Structure Example

```
my-website/
│
├── index.html
├── style.css
└── images/
```

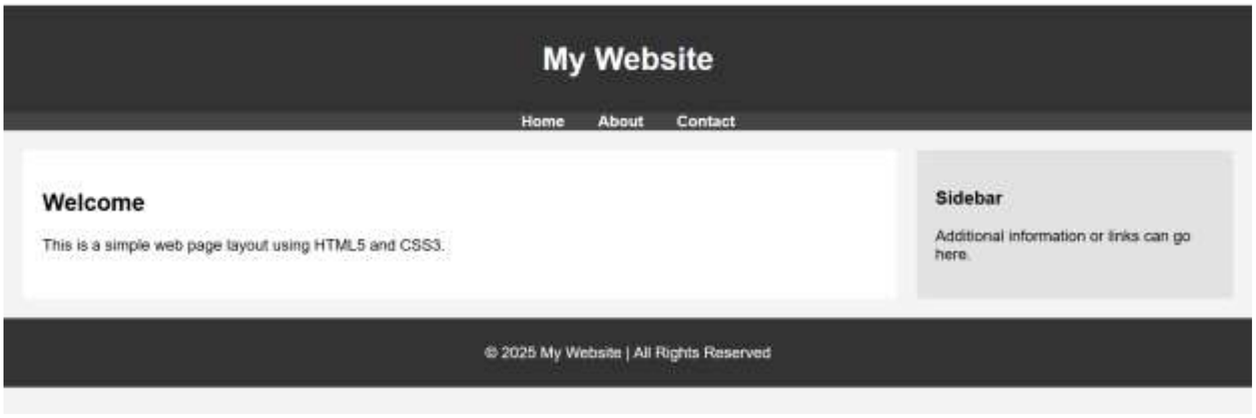
This structure keeps your project clean and organized.

2.2.4.5. Final Output (What Students Should See)

A professional-looking webpage with:

- A header and navigation bar at the top
- A main content section and sidebar in the middle
- A footer at the bottom
- Responsive behavior when resized

Laptop Screen Output:



My Website

Home About Contact

Welcome

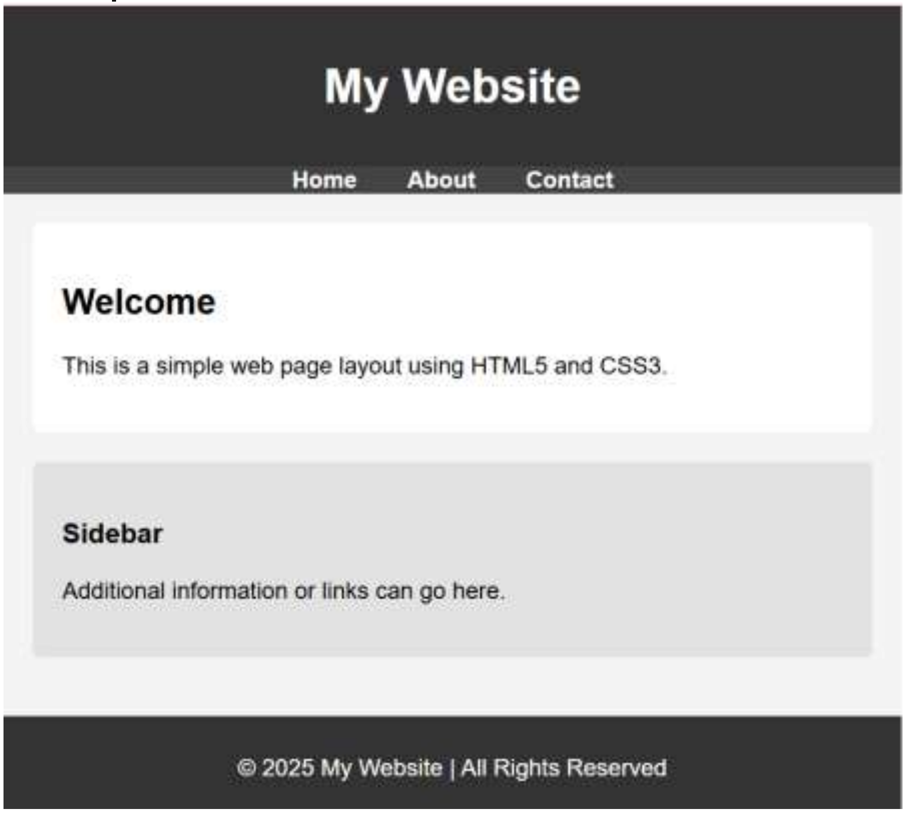
This is a simple web page layout using HTML5 and CSS3.

Sidebar

Additional information or links can go here.

© 2025 My Website | All Rights Reserved

Mobile Screen Output:



My Website

Home About Contact

Welcome

This is a simple web page layout using HTML5 and CSS3.

Sidebar

Additional information or links can go here.

© 2025 My Website | All Rights Reserved

LU2.2.5. Introduction to Git & Version Control

When developing web applications, multiple developers often work together, make frequent code changes, and experiment with new features. To manage all these changes efficiently and avoid losing work, developers use Version Control Systems (VCS).

Git is the most widely used version control system that helps developers track changes, collaborate, and maintain the complete history of their project codebase.

2.2.5.1. What is Version Control?

Version Control is a system that records changes to files over time so you can recall specific versions later.

It allows you to:

- Track changes: See what was changed, when, and by whom.
- Revert mistakes: Restore previous versions if something breaks.
- Collaborate easily: Multiple people can work on the same project without overwriting each other's work.

There are two main types of version control systems:

Type	Description	Example

Centralized VCS	A single central server stores all versions, and developers get copies from there.	SVN, CVS
Distributed VCS	Every developer has a full copy of the project, including history. Changes are shared via repositories.	Git, Mercurial

2.2.5.2. What is Git?

Git is a Distributed Version Control System created by Linus Torvalds (the creator of Linux) in 2005.
It allows you to:

- Work offline (every copy is a complete repo).
- Merge code easily between team members.
- Keep a full history of every change made.
- Integrate with cloud platforms like GitHub, GitLab, or Bitbucket.

2.2.5.3. Key Concepts in Git

Concept	Description
Repository (Repo)	A directory that contains your project files and Git history.
Commit	A snapshot of your code at a specific point in time.
Branch	A separate line of development. For example, main and feature-login.
Merge	Combining changes from one branch into another.
Clone	Copying an existing remote repository to your local computer.
Push	Uploading your local changes to the remote repository (e.g., GitHub).
Pull	Downloading and merging changes from a remote repository.

2.2.5.4. Installing Git

i. Go to https://git-scm.com/downloads.
ii. Download and install Git for your operating system (Windows, macOS, or Linux).
iii. Verify installation by running:
iv. git --version
v. Configure your identity (done once):
vi. git config --global user.name "Your Name"
vii. git config --global user.email "your@email.com"

2.2.5.5. Basic Git Workflow

Let's understand the typical Git process step by step:

Step 1: Initialize Repository

- Create a Git repo in your project folder:
    - **git init**

Step 2: Check Status

- View file changes:
    - **git status**

Step 3: Add Files

- Stage files for commit:
- git add index.html style.css
- or add all:
    - **git add**

Step 4: Commit Changes

- Save a snapshot with a meaningful message:
    - **git commit -m "Initial website layout"**

Step 5: Connect to Remote Repository

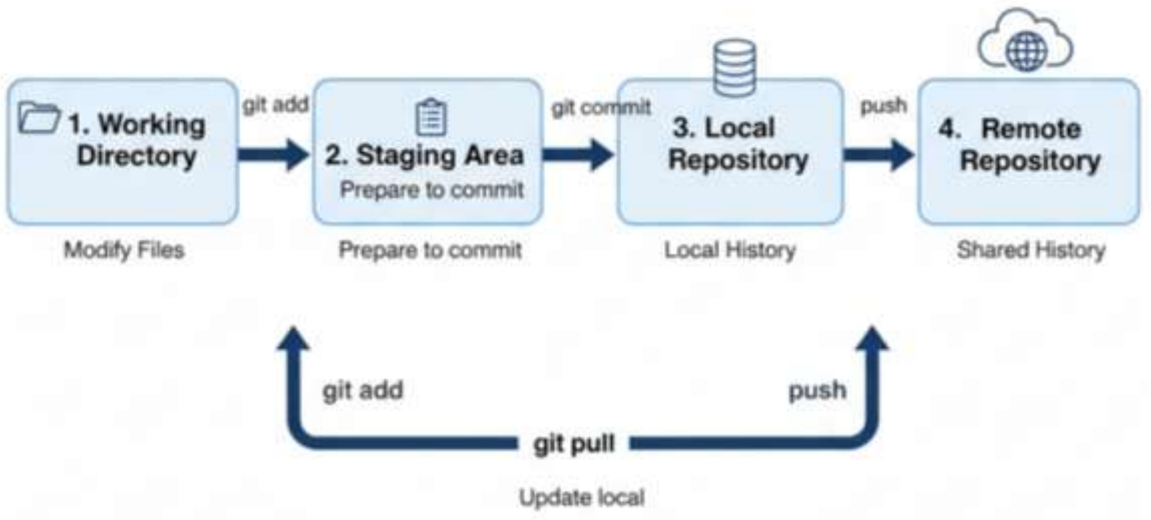
- Link your local repo with a remote GitHub repo:
    - **git remote add origin https://github.com/username/myproject.git**

Step 6: Push Changes

- Send your commits to GitHub:
    - **git push -u origin main**

Step 7: Pull Updates

- Fetch and merge changes from the remote repository:
    - **git pull origin main**



2.2.5.6. Example Folder Setup

```
myproject/
│
├── index.html
├── style.css
└── .git/
```

Once initialized, Git starts tracking changes in all project files inside this folder.

2.2.5.7. Advantages of Using Git

✅ Keeps a full history of all code changes
✅ Enables teamwork and collaboration
✅ Allows experimentation through branches
✅ Prevents data loss
✅ Supports integration with GitHub for sharing and review

2.2.5.8. GitHub Overview

GitHub is a cloud platform where developers store, manage, and share their Git repositories. It allows for:

- Team collaboration
- Code reviews and pull requests
- Issue tracking
- Project visibility and hosting

2.3. Practical Units (PUs)

PU2.3.1. Exploring the MERN Stack Components
Objective: To understand and identify the four core components of the MERN Stack and their roles in full-stack web development.
Tasks:

1. Research and document the purpose of MongoDB, Express.js, React.js, and Node.js.
2. Draw a labeled diagram showing how data flows between frontend (React), backend (Node & Express), and database (MongoDB).
3. Identify real-world examples of websites or apps built using the MERN Stack.

PU2.3.2. Setting Up the MERN Development Environment
Objective: To install and configure Node.js and Visual Studio Code for MERN stack development.
Tasks:

1. Download and install Node.js (LTS version).
2. Verify installation using node -v and npm -v.
3. Install and set up Visual Studio Code.
4. Add essential extensions (Prettier, ES7+ React Snippets, Live Server, GitLens, MongoDB for VS Code).
5. Create and run a simple app.js file displaying "Environment Setup Successful!".

PU2.3.3. Creating a Basic HTML5 Web Page
Objective: To build a basic HTML5 page using core tags and structure.
Tasks:

1. Create a new folder named web-layout.
2. In VS Code, create an index.html file.
3. Include essential HTML5 elements:
   - <!DOCTYPE html>, <html>, <head>, <title>, and <body>.
   - Add a heading, paragraph, image, and hyperlink.
4. Use semantic tags like <header>, <main>, <section>, and <footer>.
5. Validate the code in a browser.

PU2.3.4. Styling a Web Page Using CSS3

Objective: To apply basic CSS styling to enhance the appearance of a web page.

Tasks:

1. Create a new file style.css and link it with index.html.
2. Apply styles for body (background color, font), headings (color, font size), and paragraphs (line spacing, alignment).
3. Use CSS selectors (class, id, element) effectively.
4. Apply CSS box model properties (margin, padding, border).
5. Add a media query to make the layout responsive on smaller screens.

PU2.3.5. Building a Simple Web Page Layout

Objective: To design a basic webpage layout using HTML5 structure and CSS positioning.

Tasks:

1. Create sections for Header, Navigation Bar, Main Content, Sidebar, and Footer.
2. Use Flexbox or CSS Grid to arrange elements.
3. Add placeholder content and images.
4. Ensure the layout adapts to different screen sizes.

PU2.3.6. Using Git for Version Control

Objective: To understand version control principles and apply basic Git commands for managing code.

Tasks:

1. Install Git and configure username & email.
2. Initialize a repository in your project folder (git init).
3. Create a .gitignore file.
4. Add files to staging (git add .) and commit changes (git commit -m "Initial commit").
5. Create a GitHub account and push the project to a remote repository.

Module 3: JavaScript Essentials & DOM Manipulation

3.3. Introduction

JavaScript is the core programming language of the web. It enables developers to add interactivity, logic, and dynamic functionality to web pages. Unlike HTML and CSS, which define structure and style, JavaScript controls how web pages behave — from handling user input and performing calculations to updating content dynamically without reloading the page. This module provides a foundational understanding of JavaScript syntax, control structures, and data handling. Learners will explore how JavaScript interacts with HTML and CSS through the Document Object Model (DOM), enabling them to manipulate web content programmatically. By the end of this module, learners will be able to create interactive and responsive web pages — a critical step toward full-stack MERN development.

Module Objectives

By the end of this module, learners will be able to:

- Understand JavaScript syntax, data types, and operators.
- Implement control structures such as conditionals and loops.
- Use functions and events to add interactivity to web pages.
- Manipulate HTML and CSS elements dynamically using the DOM.
- Handle user input and validate form data.
- Apply basic debugging and browser console tools for error handling.

3.2. Learning Units (LUs)

LU3.2.1: JavaScript Fundamentals — Variables, Loops, and Functions

JavaScript is the core programming language that powers interactivity on the web. Understanding its fundamental building blocks — variables, loops, and functions — is essential for writing efficient, dynamic, and reusable code. These elements form the foundation for all logic-driven web development tasks, from simple animations to complex full-stack applications.

This unit introduces how data is stored, repeated operations are handled, and reusable actions are created using JavaScript. By mastering these basics, learners will be prepared to build interactive and responsive web features.

3.2.1.1. Variables

Variables are containers used to store and manage data in JavaScript. They allow developers to label and reuse values throughout the program.

Types of Variables:

- **var** – Old variable declaration (function-scoped).
- **let** – Block-scoped variable; recommended for use in modern JavaScript.
- **const** – Used for constants whose values should not change.

Example:

```
let username = "Muskan";
const maxScore = 100;
var age = 22;
console.log("User:", username);
```

Key Rules:

- Variable names must start with a letter, underscore _, or dollar sign $.
- JavaScript is case-sensitive (User ≠ user).
- Use meaningful variable names for better readability.

3.2.1.2. Data Types

JavaScript supports different data types to store and process various kinds of information.

Primitive Data Types:

- **String:** "Hello World"
- **Number:** 25, 3.14
- **Boolean:** true or false
- **Null:** Represents an intentional empty value
- **Undefined:** A variable declared but not assigned a value

Complex Data Types:

- **Array:** Stores multiple values in a single variable.
- **Object:** Stores data as key-value pairs.

Example:

```
let colors = ["red", "green", "blue"];
let student = { name: "Ali", age: 20, enrolled: true };
```

3.2.1.3. Loops

Loops allow developers to repeat a block of code multiple times without writing it repeatedly.

Common Types of Loops:

- **for loop** – Repeats a block of code a specific number of times.
```
for (let i = 1; i <= 5; i++) {
  console.log("Count:", i);
}
```
- **while loop** – Runs while a condition is true.
```
let num = 1;
while (num <= 3) {
  console.log("Number:", num);
  num++;
}
```
- **for...of loop** – Used for iterating over arrays.
```
let fruits = ["Apple", "Mango", "Orange"];
for (let fruit of fruits) {
  console.log(fruit);
}
```

Loops make it easier to process lists, perform calculations, and handle repetitive tasks efficiently.

3.2.1.4. Functions

Functions are reusable blocks of code designed to perform a specific task. They help reduce repetition, improve readability, and make programs modular.

Defining and Calling a Function:

```
function greetUser(name) {
  console.log("Hello, " + name + "!");
}
greetUser("Muskan");
```

Arrow Function (Modern Syntax):

```
const addNumbers = (a, b) => {
  return a + b;
};
console.log(addNumbers(5, 10)); // Output: 15
```

Key Concepts:

- Functions can take parameters (inputs) and return values (outputs).
- Functions can be called multiple times with different arguments.
- Arrow functions (=>) are concise and widely used in modern JavaScript frameworks like React.

3.2.1.5. Combining Concepts

Variables, loops, and functions often work together in real-world scenarios.

Example Program:

```
function printNumbers(limit) {
  for (let i = 1; i <= limit; i++) {
    console.log("Number:", i);
  }
}
let userLimit = 5;
printNumbers(userLimit);
```

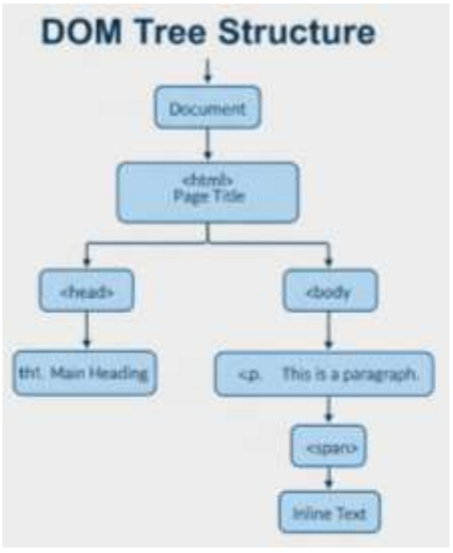
This script uses a variable (userLimit), a loop, and a function to print numbers dynamically based on user input.

3.2.1.6. Best Practices

- Always use let and const instead of var for clarity and scope control.
- Keep functions short and specific to a single task.
- Use indentation and comments for readability.
- Avoid global variables; use local scope wherever possible.
- Test small parts of code frequently using console.log() for debugging.

LU3.2.2. DOM Manipulation — Accessing and Modifying HTML Elements

The Document Object Model (DOM) is a structured representation of a web page that allows JavaScript to interact with and manipulate HTML elements dynamically. When a webpage loads, the browser converts its HTML into a tree-like structure — the DOM — where each element (like a heading, paragraph, or button) becomes a node that JavaScript can access and modify.



DOM manipulation enables developers to create interactive and responsive web pages without reloading. For example, clicking a button can change text, display an image, or hide content instantly using JavaScript.

3.2.2.1. Understanding the DOM

The DOM treats an HTML document as a tree of nodes:

- The root of the tree is the <html> element.
- Inside it are child nodes like <head> and <body>.
- Each HTML tag, attribute, and text content becomes a node in the structure.

JavaScript accesses this tree through the document object — a global object available in every web page.

Example (HTML):

```
<h1 id="title">Welcome to My Page</h1>
<p class="info">This is a short introduction paragraph.</p>
```

Example (JavaScript):

```
let heading = document.getElementById("title");
console.log(heading.innerText);  // Output: Welcome to My Page
```

3.2.2.2. Accessing HTML Elements

JavaScript provides several methods to access elements within the DOM:

Method	Description	Example
getElementById()	Selects a single element by its ID	document.getElementById("title")
getElementsByClassName()	Returns all elements with a specific class (HTMLCollection)	document.getElementsByClassName("info")
getElementsByTagName()	Selects elements by their tag name	document.getElementsByTagName("p")
querySelector()	Selects the first matching element using a CSS selector	document.querySelector(".info")
querySelectorAll()	Selects all matching elements (NodeList)	document.querySelectorAll("p")

Example:

```
let paragraphs = document.querySelectorAll("p");
console.log("Total paragraphs:", paragraphs.length);
```

3.2.2.3. Modifying HTML Content

Once elements are accessed, their content and properties can be changed dynamically.

Common Properties and Methods:

- innerText — changes or retrieves visible text.
- innerHTML — changes or retrieves HTML inside an element.
- style — changes inline CSS styles.
- setAttribute() — changes attributes (like src, href, etc.).
- classList — adds or removes CSS classes.

Example:

```
let heading = document.getElementById("title");
heading.innerText = "Hello, JavaScript Learner!";
heading.style.color = "blue";
heading.style.fontSize = "28px";
```

3.2.2.4. Creating and Adding New Elements

JavaScript can dynamically create and insert new HTML elements into the DOM.

Example:

```
let newPara = document.createElement("p");
newPara.innerText = "This paragraph was added using JavaScript!";
document.body.appendChild(newPara);
```

- createElement() – creates a new element.
- appendChild() – adds it to the document.
- remove() – deletes an element from the page.

3.2.2.5. Real-World Example

HTML:

```
<h2 id="heading">Click the Button</h2>
<button onclick="changeText()">Change Text</button>
```

JavaScript:

```
function changeText() {
  let element = document.getElementById("heading");
  element.innerText = "You just changed the text!";
  element.style.color = "green";
}
```

This example shows how user interaction (a click) triggers a change in both content and style through DOM manipulation.

3.2.2.6. Best Practices

- Use meaningful IDs and class names for easy element selection.
- Avoid using innerHTML unnecessarily (for security reasons).
- Keep JavaScript separate from HTML using external .js files.
- Always test in the browser console before implementing major changes.
- Use querySelector for flexibility with CSS-like selectors.

LU3.2.3. Event Handling and Basic Animations

Event handling and animations make web pages interactive and lively. In JavaScript, an event is any action that occurs in the browser — such as a mouse click, key press, or page load. Event handling means writing code that responds to these user actions. Animations, on the other hand, are visual effects that make elements move, change, or transition smoothly over time.

Together, events and animations help create dynamic, user-friendly websites that react to users in real time.

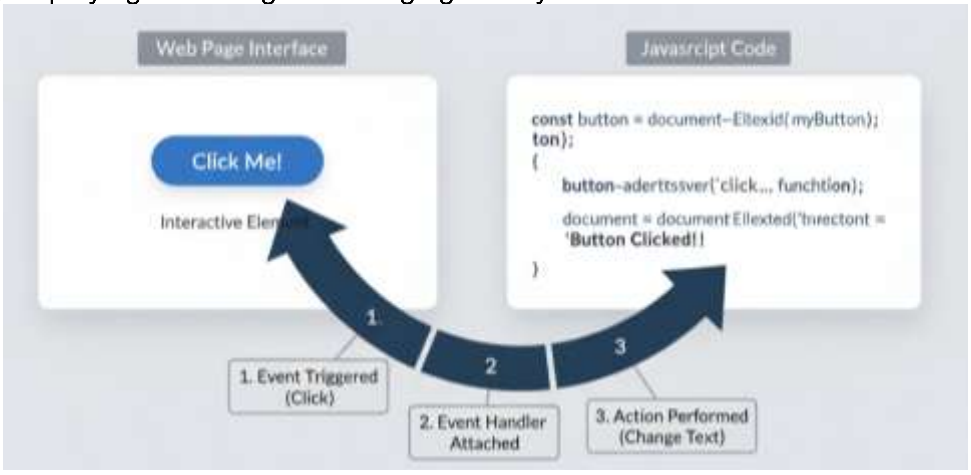
3.2.3.1. What is an Event?

An event is a signal that something has happened on a web page.

Examples include:

- Mouse events: click, dblclick, mouseover, mouseout
- Keyboard events: keydown, keyup
- Form events: submit, change, focus
- Window events: load, resize, scroll

When an event occurs, JavaScript can execute a function that performs an action — for example, displaying a message or changing the style of an element.


Web Page Interface
Click Me!
Interactive Element
Javasrcipt Code
1
2
3
1. Event Triggered (Click)
2. Event Handler Attached
3. Action Performed (Change Text)

```
const button = document–Eltexid(myButton);
ton);
{
    button-aderttssver('click... funchtion);
    document = document Ellexted('Inrectont =
    'Button Clicked!!
}
```

3.2.3.2. Handling Events in JavaScript

A. Inline Event Handling (Basic)

You can attach an event directly in the HTML element using attributes like onclick.

Example:

```
<button onclick="showMessage()">Click Me</button>

<script>
function showMessage() {
  alert("Button was clicked!");
}
```

```
</script>
```
Simple and easy to use for small examples.
Not recommended for large projects (mixes HTML and JS).

B. Event Listeners (Preferred Method)

Modern JavaScript uses event listeners for cleaner code and flexibility.

Example:
```
<button id="myButton">Click Me</button>
<script>
let button = document.getElementById("myButton");
button.addEventListener("click", function() {
  alert("Event handled with addEventListener!");
});
</script>
```

Advantages:
- Keeps HTML clean.
- Allows multiple functions to respond to the same event.
- Easier to manage complex interactions.

3.2.3.3. Common Event Examples

Mouse Events Example:
```
<button id="hoverBtn">Hover Over Me</button>

<script>
let btn = document.getElementById("hoverBtn");
btn.addEventListener("mouseover", function() {
  btn.style.backgroundColor = "lightgreen";
});
btn.addEventListener("mouseout", function() {
  btn.style.backgroundColor = "";
});
</script>
```

Keyboard Event Example:
```
<input type="text" id="nameInput" placeholder="Type your name">
<script>
document.getElementById("nameInput").addEventListener("keyup", function() {
  console.log("You pressed a key!");
});
</script>
```

3.2.3.4. Introduction to Basic Animations

Animations make changes appear smooth and gradual instead of instant. They can be done using:
- **CSS Transitions** (simple animations)
- **JavaScript-based Animations** (for dynamic control)

A. CSS Transition Example
```
<style>
#box {
  width: 100px;
  height: 100px;
  background-color: coral;
  transition: transform 0.5s ease;
}
#box:hover {
  transform: scale(1.2);
}
</style>

<div id="box"></div>
```

When you hover the box, it smoothly enlarges using CSS transitions.

B. JavaScript Animation Example (Using setInterval)

```
<div id="animateBox" style="width:100px; height:100px; background:blue; position:absolute;"></div>

<script>
let box = document.getElementById("animateBox");
let position = 0;

function moveBox() {
  if (position < 300) {
    position++;
    box.style.left = position + "px";
  } else {
    clearInterval(timer);
  }
}
let timer = setInterval(moveBox, 10);
</script>
```

This code moves a blue box smoothly across the screen using JavaScript.

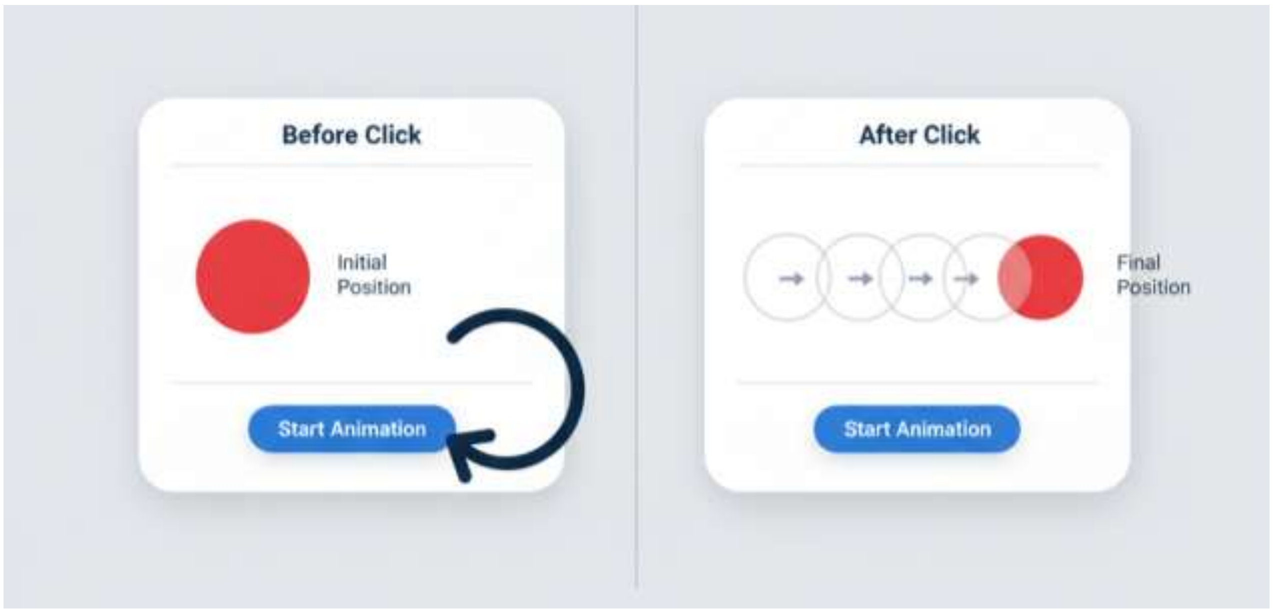
3.2.3.5. Combining Events and Animations

You can trigger animations with events for example, start moving a box when a button is clicked.

Example:

```
<button id="startBtn">Start Animation</button>
<div id="ball" style="width:50px; height:50px; background:red; border-radius:50%; position:absolute;"></div>

<script>
document.getElementById("startBtn").addEventListener("click", function() {
  let ball = document.getElementById("ball");
  let pos = 0;
  let move = setInterval(function() {
    if (pos >= 300) clearInterval(move);
    else {
      pos += 2;
      ball.style.left = pos + "px";
    }
  }, 10);
});
</script>
```

This example shows event-driven animation the animation starts only when the user clicks the button.



LU3.2.4. Introduction to ES6 Features: Arrow Functions, Let/Const, Template Literals
ECMAScript 2015, also known as ES6, introduced several new features to make JavaScript more powerful, readable, and efficient.
Before ES6, JavaScript had limitations such as complex syntax, function scoping issues, and messy string concatenation
With ES6, developers can write modern, maintainable, and concise code — essential for building large-scale web applications like those in the MERN stack.
In this unit, we'll explore three key ES6 features:

1. **Arrow Functions**
2. **Let and Const Declarations**
3. **Template Literals**

3.2.4.1. Arrow Functions (=>)

What are Arrow Functions?
Arrow functions provide a shorter syntax for writing functions in JavaScript. They also automatically handle this keyword in a more predictable way, which is useful in event handling and callbacks.
Example (Traditional vs Arrow Function)
Traditional Function:

```
function greet(name) {
  return "Hello, " + name + "!";
}
```

Arrow Function:

```
const greet = (name) => `Hello, ${name}!`;
```

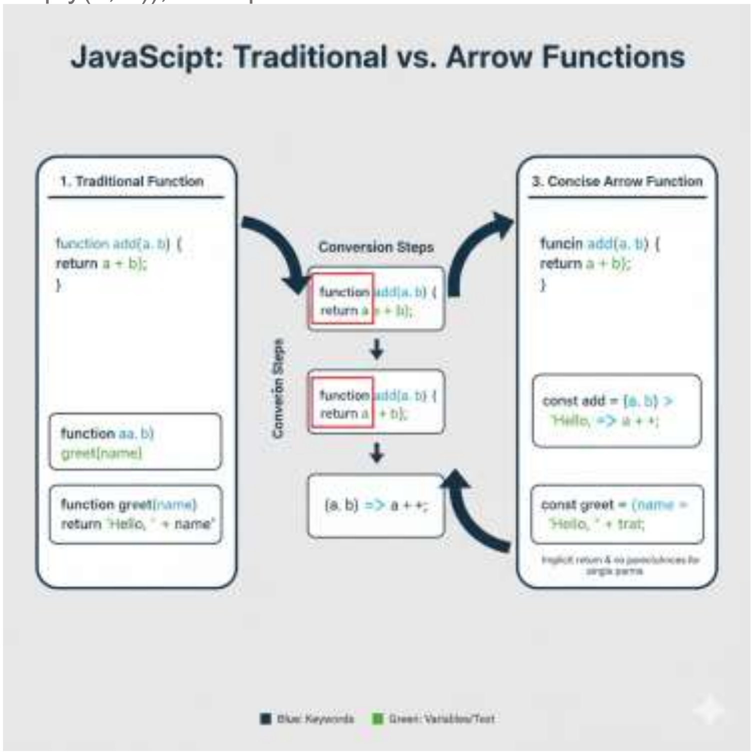
- Shorter and cleaner syntax
- Automatically returns the value if the function body has only one line
- No need to bind this manually in most cases

Examples:
1. With Parameters:

```
const add = (a, b) => a + b;
console.log(add(3, 4)); // Output: 7
```

2. No Parameters:

```
const sayHi = () => console.log("Hi there!");
```

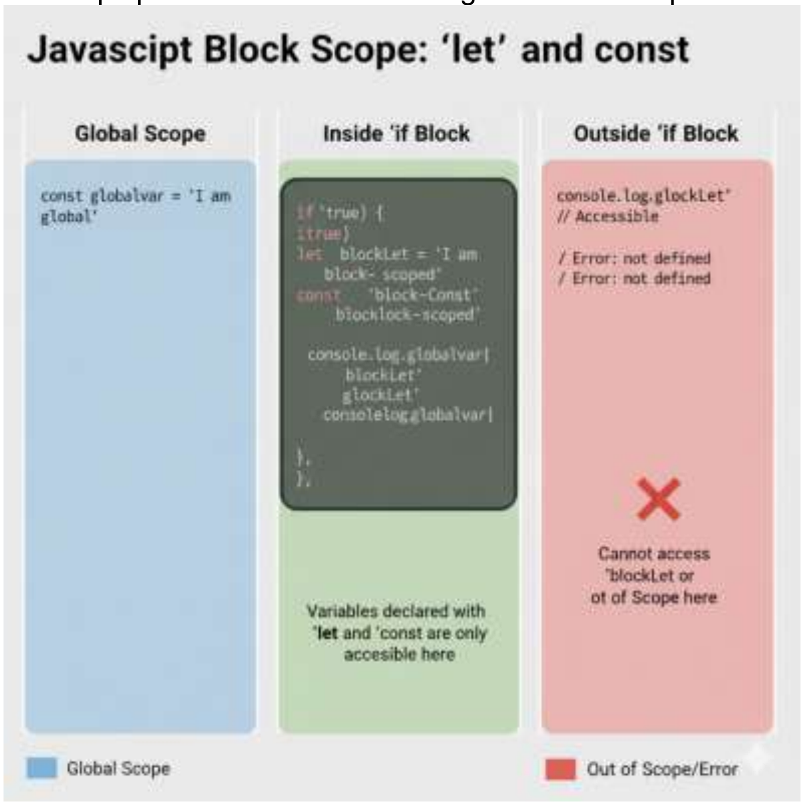
```
        sayHi();
```
3. Multi-line Function:
```
        const multiply = (x, y) => {
          const result = x * y;
          return result;
        };
        console.log(multiply(4, 5)); // Output: 20
```


3.2.4.2. Let and Const (Block Scope Variables)

Before ES6, JavaScript used only var for variable declarations, which had function scope, often leading to unexpected behavior.
ES6 introduced let and const to solve these problems.

Keyword	Scope Type	Reassignment Allowed	Typical Use
var	Function Scope	✅ Yes	Legacy code
let	Block Scope	✅ Yes	Variables that change
const	Block Scope	❌ No	Fixed values or constants

Example:
```
let count = 10;
const name = "Arham";
count = 15;   // ☑ Allowed
// name = "Ali"; ✖ Error — cannot reassign const

console.log(count); // 15
console.log(name);  // Arham
```
Block Scope Example:
```
if (true) {
  let x = 5;
  const y = 10;
  var z = 15;
}
console.log(z); // ☑ Accessible (var is function-scoped)
console.log(x); // ✖ Error (let is block-scoped)
console.log(y); // ❌ Error (const is block-scoped)
```
Using let and const helps prevent accidental reassignments and improves code safety.


Javascipt Block Scope: ‘let’ and const
Global Scope
Inside ‘if Block
Outside ‘if Block
Variables declared with ‘let and ‘const are only accesible here
Cannot access ‘blockLet or ot of Scope here
🟦 Global Scope
🟥 Out of Scope/Error

```
const globalvar = 'I am
global'
```

```
if 'true) {
 itrue)
let  blockLet = 'I am
    block- scoped'
const    'block-Const'
      blocklock-scoped'

  console.log.globalvar|
      blockLet'
     glockLet'
   consolelog.globalvar|

},
};
```

```
console.log.glockLet'
// Accessible

/ Error: not defined
/ Error: not defined
```

3.2.4.3. Template Literals (Backticks ``)

Template literals allow embedding variables and expressions directly into strings, using backticks (``) instead of quotes.

Old Way (String Concatenation):

```
        let name = "Sara";
        let message = "Hello, " + name + "! Welcome to JavaScript.";
```

New Way (Template Literals):

```
let name = "Sara";
let message = `Hello, ${name}! Welcome to JavaScript.`;
console.log(message);
```

- Easier to read
- Supports multi-line strings
- Allows inserting expressions directly inside ${ }

Example:

```
        let a = 5;
        let b = 10;
        console.log(`The sum of ${a} and ${b} is ${a + b}.`);
```

Output:

```
The sum of 5 and 10 is 15.
```

3.2.4.4. Combining All Features Together

```
const greetUser = (firstName, lastName) => {
  const fullName = `${firstName} ${lastName}`;
  return `Welcome, ${fullName}!`;
};

console.log(greetUser("Ali", "Khan"));
```

Uses:

- Arrow function
- const for constants

- Template literal for message formatting

3.3. Practical Units (PUs)

PU3.3.1. Practicing JavaScript Fundamentals
Objective: To understand and apply basic JavaScript concepts such as variables, operators, loops, and functions.
Tasks:

- **i.** Create a new JavaScript file basics.js and practice:
    - Declaring variables using let, const, and var.
    - Performing basic arithmetic operations.
- **ii.** Write a program that:
    - Takes user input for two numbers and prints their sum, difference, and product.
- **iii.** Practice different loops:
    - for, while, and do...while loops to print numbers 1–10.
- **iv.** Create simple reusable functions:
    - Example: function greetUser(name) → returns "Hello, [name]!"
    - Example: function calculateArea(radius) → returns circle area.
- **v.** Test all outputs in the browser console.

PU3.3.2. Accessing and Modifying the DOM
Objective: To understand how JavaScript interacts with HTML elements using the Document Object Model (DOM).
Tasks:

- **i.** Create an HTML file with:
    - A heading <h1 id="mainHeading">Welcome</h1>
    - A paragraph <p id="info">This is a JavaScript DOM example.</p>
    - A button <button id="changeText">Click Me</button>
- **ii.** In script.js, use JavaScript to:
    - Access elements by ID, class, and tag name.
    - Change text content and style properties.
    - Example:
    - document.getElementById("mainHeading").innerText = "Hello, DOM!";
    - document.getElementById("info").style.color = "blue";
- **iii.** Add a new HTML element dynamically using createElement() and appendChild().
- **iv.** Practice removing and updating elements.

PU3.3.3. Event Handling and Basic Animations
Objective: To learn how to respond to user actions and create simple animations using JavaScript.
Tasks:

- **i.** Create an HTML page with a button and an image.
- **ii.** Use addEventListener() to handle events:
    - click, mouseover, and mouseout.
    - Example:
    - document.getElementById("btn").addEventListener("click", () => {
    -   alert("Button clicked!");
    - });
- **iii.** Change image size or color on button click.
- **iv.** Implement a simple animation using setInterval() or CSS transitions triggered by JavaScript.
    - Example: Moving a box across the screen.

PU3.3.4. Exploring ES6 Features
Objective: To practice writing modern JavaScript using ES6 syntax and features.
Tasks:

- **i.** Create a file es6_practice.js.
- **ii.** Practice variable declaration using let and const.
- **iii.** Write a few arrow functions for mathematical operations like addition, subtraction, or greetings.
- **iv.** Use template literals to dynamically generate strings:
- **v.** const name = "Ali";
- **vi.** const age = 21;
- **vii.** console.log(`My name is ${name} and I am ${age} years old.`);
- **viii.** Combine all concepts to build a small output message generator.

PU3.3.5. Mini Project — Interactive Web Page
Objective: To combine all learned concepts (JavaScript, DOM, Events, and ES6) in one mini-project.
Tasks:

- **i.** Create a simple interactive web page (e.g., a digital greeting card, quiz app, or counter).
- **ii.** Use DOM to update text and images dynamically.
- **iii.** Handle at least two user events (button click, hover, etc.).
- **iv.** Apply ES6 features (let, const, arrow functions, template literals`) in the script.
- **v.** Style with basic CSS and test functionality.

Module 4: React.js Fundamentals

4.1. Introduction

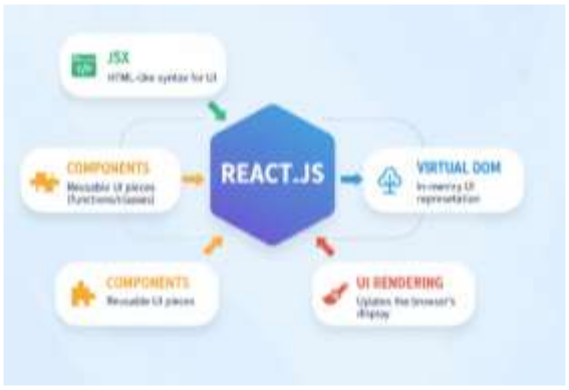
React.js is a popular JavaScript library used for building dynamic and interactive user interfaces, especially single-page applications (SPAs). Developed by Facebook, React helps developers create reusable UI components that make web applications fast, maintainable, and efficient.
It uses a Virtual DOM to update only the parts of the page that change, improving performance compared to traditional JavaScript rendering.
This module introduces students to the core concepts of React, including setting up a project, understanding components, props, state, and event handling. By the end of this module, learners will be able to create small interactive applications using React.

4.2. Learning Units (LUs)

LU4.2.1. Introduction to React.js and JSX

React.js is a powerful JavaScript library developed by Facebook for building fast, interactive, and reusable user interfaces (UIs) — especially for single-page applications (SPAs). Instead of reloading entire pages, react updates only the specific parts of the UI that change, improving performance and user experience.
React follows a component-based architecture, where the UI is divided into small, independent pieces called components. Each component manages its own structure, style, and logic, making development organized and scalable.

4.2.1.1. Why React.js?



React revolutionized web development by introducing:

- **Reusable Components** – Build once, reuse anywhere.
- **Virtual DOM** – Efficiently updates only changed elements instead of re-rendering the whole page.
- **Declarative Syntax** – Developers describe *what* the UI should look like, not *how* to change it step-by-step.
- **Strong Community & Ecosystem** – Supported by numerous libraries, developer tools, and frameworks (like Next.js).

React enables faster rendering, cleaner code, and easier debugging compared to traditional JavaScript or jQuery.

3.2.1.2. Understanding JSX (JavaScript XML)

JSX is a syntax extension for JavaScript that allows developers to write HTML-like code inside JavaScript files.

It combines the structure of HTML with the power of JavaScript, making UI development more intuitive.

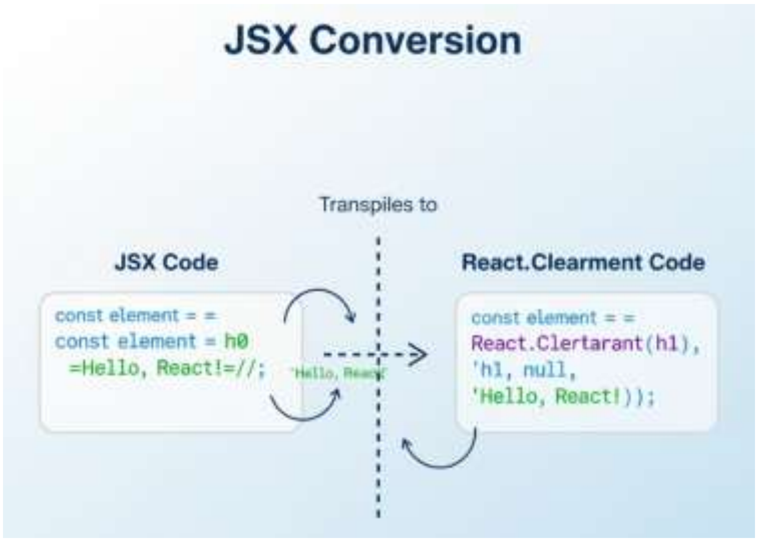
Example:

function Welcome() {
  return <h1>Hello, React!</h1>;
}

This looks like HTML but is actually JSX.

Behind the scenes, JSX is compiled into JavaScript using a tool like **Babel**, turning it into:

React.createElement("h1", null, "Hello, React!");



```
const element = =
const element = h0
  =Hello, React!=//;
```

```
const element = =
React.Clertarant(h1),
'h1, null,
'Hello, React!));
```

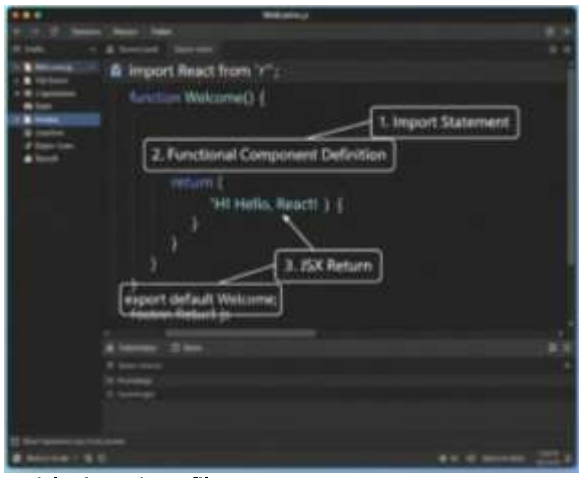
Benefits of JSX:

- Easier to visualize UI structure
- Reduces syntax errors
- Integrates JavaScript logic directly into UI code

4.2.1.3. React Component Example

A simple React component can look like this:

import React from "react";
function Greeting() {
  const name = "Ali";
  return <h2>Welcome, {name}!</h2>;
}
export default Greeting;



```
import React from 'r';
function Welcome() {
    return (
        'HI Hello, React! ) {
        }
      }
    }
export default Welcome;
```

Here:

- import React – Brings React library features.
- Greeting() – Defines a functional component.
- {name} – Embeds a JavaScript variable inside JSX.
- export default – Makes the component reusable in other files.

4.2.1.4. How React Renders the UI

i. JSX code is written inside components.
ii. Babel compiles JSX → JavaScript.
iii. React creates a Virtual DOM copy of the UI.
iv. When data changes, React updates only the changed parts in the real DOM, improving efficiency.

LU4.2.2. Functional Components and State Management

In React.js, functional components are the foundation of modern web applications. They are simple JavaScript functions that return JSX to describe what should appear on the screen. Combined with state management, these components can handle dynamic data, user input, and interactive behaviors efficiently.

Functional components make code cleaner, easier to debug, and more reusable than older class-based components.

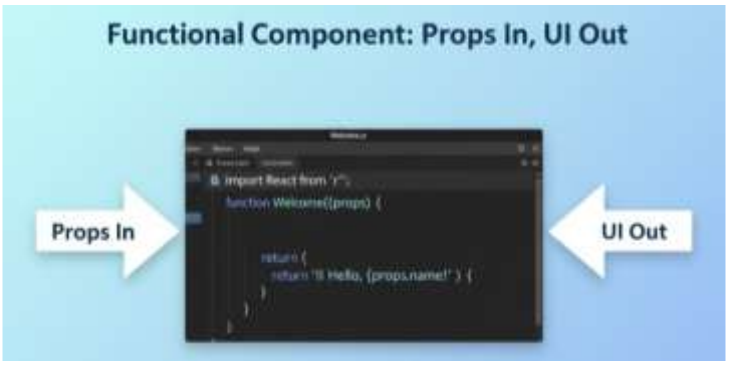
4.2.2.1. What are Functional Components?

Functional components are stateless or stateful functions that take input data (called props) and return JSX to render UI elements.

Example:

function Welcome(props) {

```
  return <h1>Hello, {props.name}!</h1>;
}
```



```
import React from 'r';
function Welcome(props) {

    return (
      return 'Hello, {props.name!' ) {
    }
  }
}
```

This component:

- Accepts data via props
- Returns JSX
- Renders output dynamically

You can reuse it anywhere in your app like this:

```
<Welcome name="Sara" />
<Welcome name="Ali" />
```

Each call renders a personalized greeting.

4.2.2.2. Introducing State in React

While props allow data to be passed into a component, state allows a component to manage its own internal data — data that can change over time.

React uses the useState() Hook to add state to functional components.

Example:

```
import React, { useState } from "react";
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
export default Counter;
```

Explanation:

- useState(0) initializes a state variable count with an initial value of 0.
- setCount is a function that updates the value of count.
- When the button is clicked, setCount increases the count and React automatically re-renders the component.

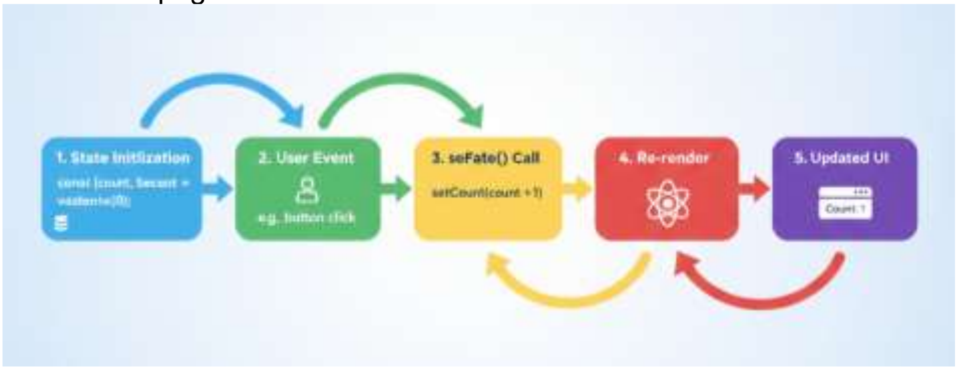
4.2.2.3. Understanding the useState Hook

The syntax of useState looks like this:

```
const [stateVariable, setStateFunction] = useState(initialValue);
```

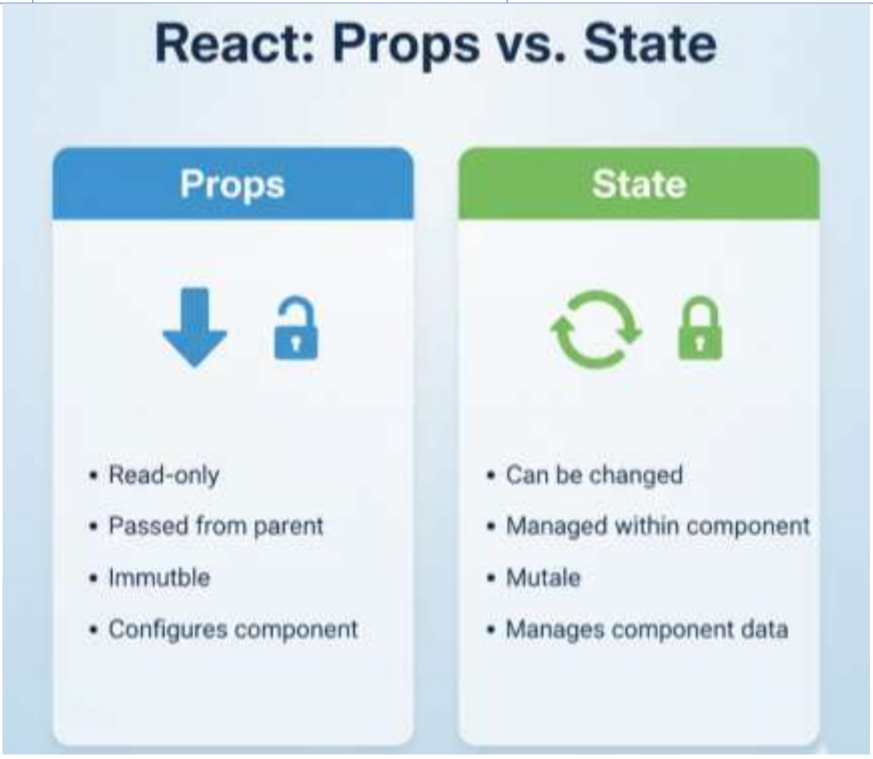
- **stateVariable:** stores the current value
- **setStateFunction:** updates the value
- **initialValue:** starting point for the state

Each time the state changes, React efficiently updates only the parts of the UI that depend on it — not the entire page.



4.2.2.4. Difference Between Props and State

Feature	Props	State
Definition	Data passed from parent to child component	Data managed inside the component
Mutability	Read-only	Mutable (can change over time)
Purpose	To make components reusable and dynamic	To handle data that changes during user interaction
Example Use	User name, image, or color	Counter value, form input, toggle state



4.2.2.5. Combining Props and State
In real-world applications, React components often use both props and state together. For example, a user profile component might receive a name via props but manage whether the profile is expanded or collapsed through state.

LU4.2.3. Props, Event Handling, and Conditional Rendering
React applications become powerful and interactive when they can exchange data, respond to user actions, and display content dynamically.
Three essential concepts enable this functionality:

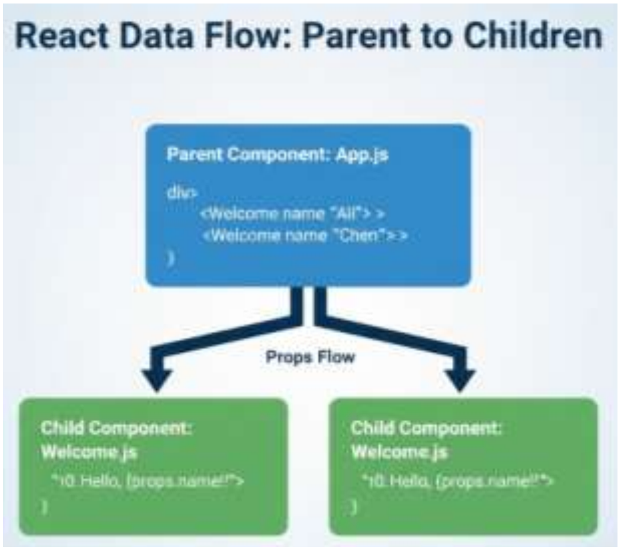
- **Props** – for passing data between components
- **Event Handling** – for responding to user interactions
- **Conditional Rendering** – for showing or hiding UI elements based on conditions

Mastering these allows developers to build truly dynamic, responsive web applications.

4.2.3.1. Understanding Props
Props (short for *properties*) are used to send data from a parent component to a child component. They make components reusable and dynamic.

Example:

```
function Welcome(props) {
  return <h2>Welcome, {props.name}!</h2>;
}

function App() {
  return (
    <div>
      <Welcome name="Muskan" />
      <Welcome name="Ali" />
    </div>
  );
}
```


React Data Flow: Parent to Children
Parent Component: App.js
div>
<Welcome name "Ali"> >
<Welcome name "Chen">>
}
Props Flow
Child Component: Welcome.js
"10 Hello, {props.name!">
}
Child Component: Welcome.js
"10 Hello, {props.name!">
}

Explanation:

- name is a prop passed from the App component to Welcome.
- Each Welcome component receives a different value and renders it dynamically.
- Props are read-only, meaning child components cannot modify them.

4.2.3.2. Event Handling in React

Event handling allows React components to respond to user actions such as clicks, typing, or mouse movements.

React uses camelCase syntax for events (e.g., onClick, onChange) and passes functions as event handlers.

Example:

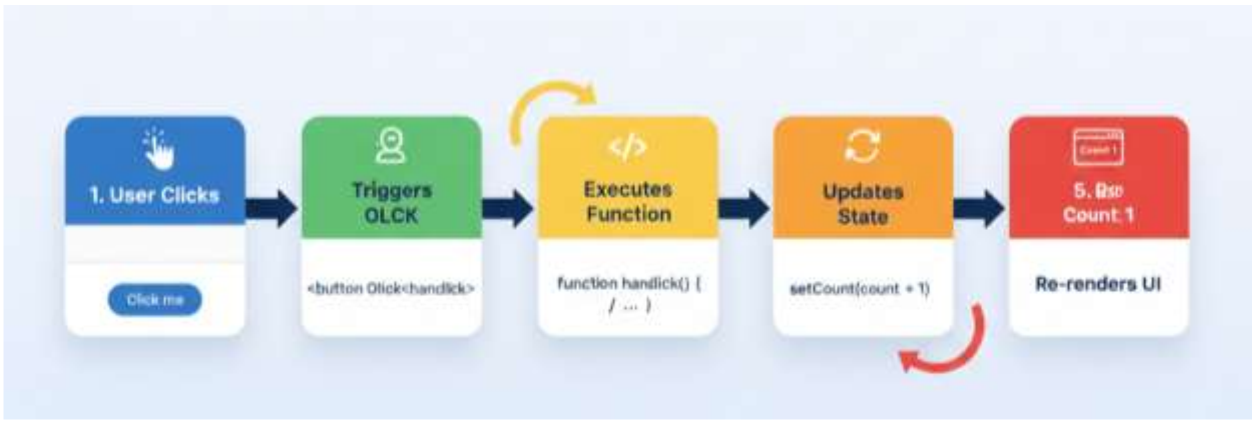
```
function ClickButton() {
  function handleClick() {
    alert("Button Clicked!");
  }
  return <button onClick={handleClick}>Click Me</button>;
}
```

Explanation:

- The onClick event triggers handleClick() when the button is pressed.
- Event handlers can also update state to change the UI dynamically.

With State Example:

```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <h3>Count: {count}</h3>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

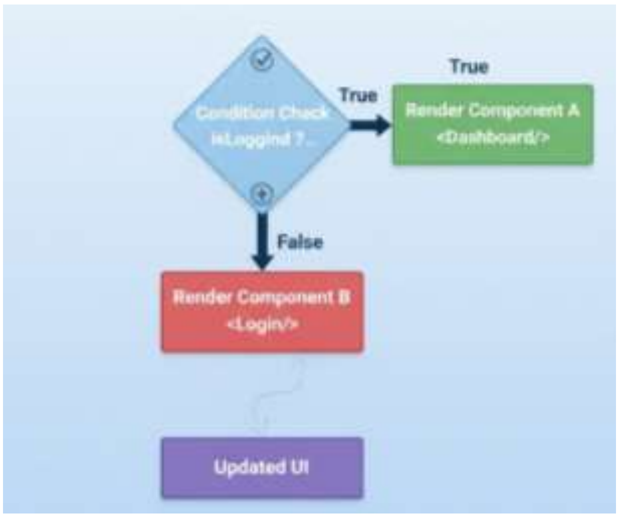


4.2.3.3. Conditional Rendering

Conditional rendering in React means showing different UI elements based on conditions (like state values or props).

Example 1: Using if-else

```
function Greeting(props) {
  if (props.isLoggedIn) {
    return <h3>Welcome Back!</h3>;
  }
  return <h3>Please Log In</h3>;
}
```



Example 2: Using Ternary Operator

```
{isLoggedIn ? <Dashboard /> : <LoginPage />}
```

Example 3: Using Logical AND (&&)

```
{notifications.length > 0 && <p>You have new notifications!</p>}
```

These techniques allow developers to display elements only when needed, keeping the interface responsive and clean.

4.2.3.4. Combining Props, Events, and Conditional Rendering

These three concepts often work together.

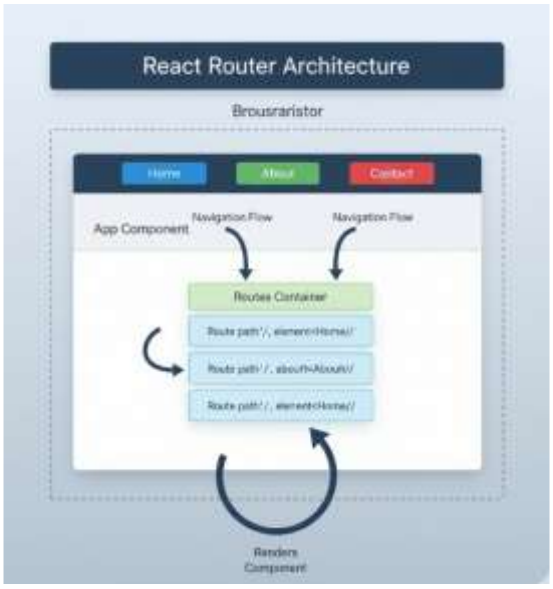
For instance, in a login component:

- Props can send user data from a parent component.
- Event handlers can process login actions.
- Conditional rendering can display messages or dashboard views depending on the login state.

This integration makes applications interactive, modular, and user-friendly.

LU4.2.4. React Router Basics for Navigation

In traditional websites, navigating between pages requires a full page reload. However, React uses a Single Page Application (SPA) model; meaning all content is loaded once, and only parts of the page update when a user interacts with it.
To manage this navigation efficiently, react developers use a library called React Router.
React Router allows your React app to have multiple views (or pages) while still maintaining the SPA behavior, meaning no reloading occurs — only the visible content changes dynamically.


React Router Architecture
Brousraristor
Home
About
Contact
App Component
Navigation Flow
Navigation Flow
Routes Container
Route path'/, element<Home/
Route path'/, about<About/
Route path'/, element<Home/
Renders Component

Key Concepts

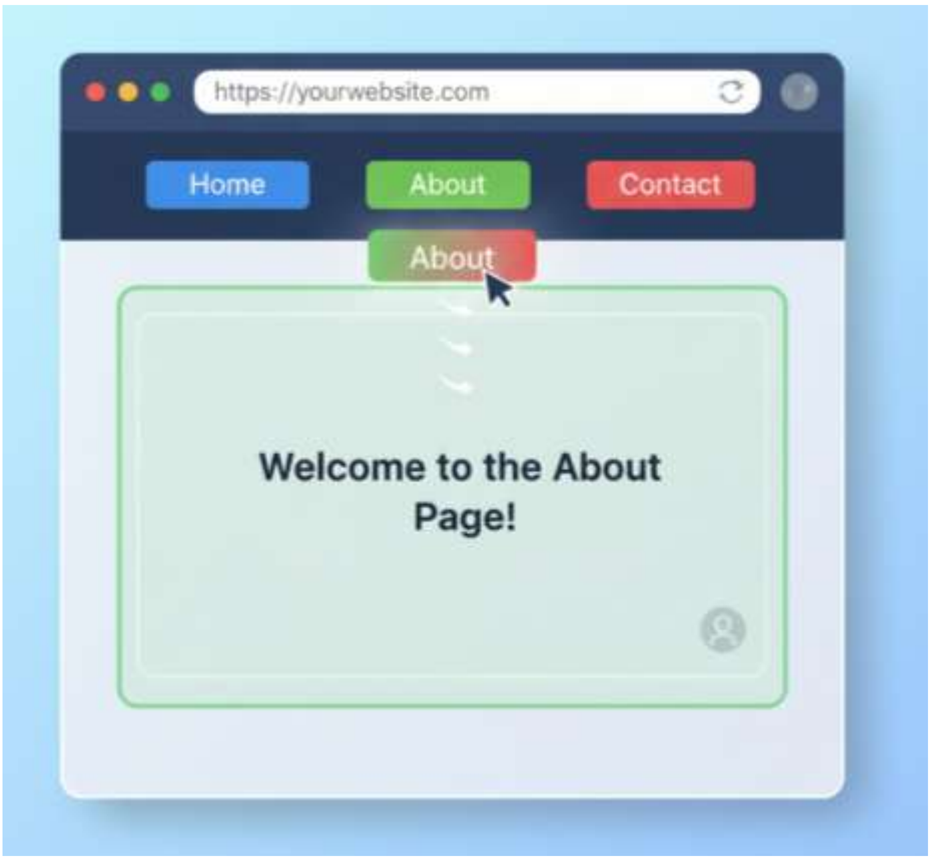
i. **BrowserRouter**
   - It acts as the main container for all your routes.
   - It uses the browser's history API to keep your UI in sync with the URL.
   - Every React Router setup starts by wrapping your main component inside a <BrowserRouter>.

```
import { BrowserRouter } from "react-router-dom";

function App() {
  return (
    <BrowserRouter>
      {/* Your routes will go here */}
    </BrowserRouter>
  );
}
```

ii. **Routes and Route**
   - The <Routes> component is a container for all the routes in your app.
   - Each <Route> defines a specific path and the component that should render for that path.

```
import { Routes, Route } from "react-router-dom";
import Home from "./Home";
import About from "./About";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
      </Routes>
    </BrowserRouter>
  );
}
```

Here:
   - When the URL is /, the Home component appears.
   - When the URL is /about, the About component appears.



iii. Link and NavLink

- To navigate between pages without refreshing, React Router provides <Link> and <NavLink>.
- These work like HTML <a> tags but prevent full reloads.
- NavLink is often used for navigation bars because it can apply an active style to the current link.

```
import { Link, NavLink } from "react-router-dom";

function Navbar() {
  return (
    <nav>
      <NavLink to="/" className="nav-item">Home</NavLink>
      <NavLink to="/about" className="nav-item">About</NavLink>
      <NavLink to="/contact" className="nav-item">Contact</NavLink>
    </nav>
  );
}
```

iv. Putting It All Together

```
import React from "react";
import { BrowserRouter as Router, Routes, Route, Link } from "react-router-dom";

function Home() {
  return <h2>Welcome to the Home Page</h2>;
}

function About() {
  return <h2>About Us Section</h2>;
}
```

```
function Contact() {
  return <h2>Contact Page</h2>;
}

function App() {
  return (
    <Router>
      <nav>
        <Link to="/">Home</Link> |{" "}
        <Link to="/about">About</Link> |{" "}
        <Link to="/contact">Contact</Link>
      </nav>

      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </Router>
  );
}
export default App;
```

When a user clicks on “About,” only the About component appears — the rest of the page remains unchanged.

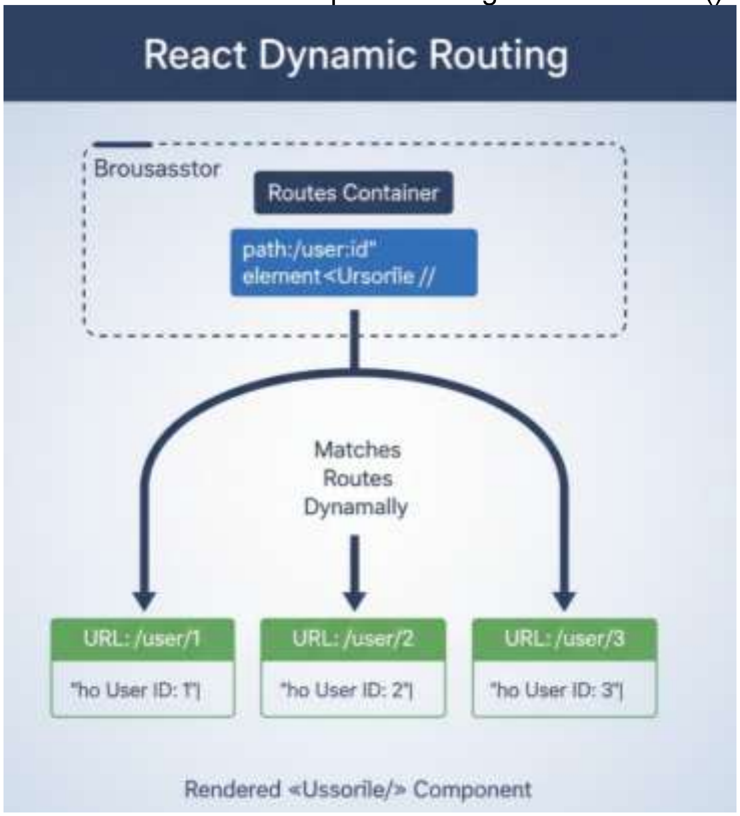
This makes the app faster and provides a smoother user experience.

v. Dynamic Routes (Advanced Concept)

Sometimes, you may want to display content based on a variable, such as a user’s ID or product name.

```
<Route path="/user/:id" element={<UserProfile />} />
```

- The :id part is a route parameter.
- It can be accessed inside the component using the useParams() hook.\



4.3. Practical Units (PUs)

PU4.1. Setting Up and Running a React Project
Objective: Learn how to create and run a React application using Create React App.
Tasks:

- Install Node.js and verify setup using node -v and npm -v.
- Create a new React project using:
- npx create-react-app my-first-react-app
- Open the project in VS Code and explore the folder structure.
- Run the app using npm start and observe the default React page.
- Replace default content in App.js with your name and a greeting message.

Outcome: Students understand how to initialize and launch a React development environment.

PU4.2. Creating Functional Components and Using JSX
Objective: Understand JSX syntax and create reusable functional components.
Tasks:

- Create a new file Header.js and define a simple functional component:
```
function Header() {
  return <h1>Welcome to My React App</h1>;
}
export default Header;
```
- Import and render it in App.js.
- Add multiple components (e.g., Footer.js, Content.js) to structure the page layout.

Outcome: Students can create and combine multiple components using JSX.

PU4.3. Managing State Using useState Hook
Objective: Learn how to manage and update component data dynamically.
Tasks:

- Create a Counter.js component.
- Use the useState hook to track and update a counter value:
```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <h2>Counter: {count}</h2>
      <button onClick={() => setCount(count + 1)}>Increase</button>
      <button onClick={() => setCount(count - 1)}>Decrease</button>
    </div>
  );
}
```
- export default Counter;
- Import it in App.js and test interaction.

Outcome: Students can maintain and update state in React components.

PU4.4. Props, Events, and Conditional Rendering
Objective: Understand how to pass data between components and handle user interactions.
Tasks:

- Create a UserCard.js component that receives user info through props.

- Use conditional rendering to display messages like “Active” or “Inactive” based on status.
- Add an event (e.g., button click) to toggle status.

Outcome: Students gain experience in using props and handling events interactively.

PU4.5. Implementing Navigation with React Router
Objective: Learn how to build a multi-page React app using React Router.
Tasks:

- Install React Router:
    - npm install react-router-dom
- Create components: Home.js, About.js, and Contact.js.
- Set up routing in App.js:

```
import { BrowserRouter, Routes, Route, Link } from "react-router-dom";
import Home from "./Home";
import About from "./About";
import Contact from "./Contact";

function App() {
  return (
    <BrowserRouter>
      <nav>
        <Link to="/">Home</Link> | <Link to="/about">About</Link> | <Link
to="/contact">Contact</Link>
      </nav>
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
      </Routes>
    </BrowserRouter>
  );
}
export default App;
```

- Test navigation between pages in the browser.

Outcome: Students can implement basic navigation across multiple components using React Router.

PU4.6. Using useEffect for Side Effects
Objective: Introduce side effects and data fetching in React.
Tasks:

- Create a component Users.js.
- Use useEffect to fetch dummy user data from an API like:

```
import React, { useState, useEffect } from "react";

function Users() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    fetch("https://jsonplaceholder.typicode.com/users")
      .then(res => res.json())
      .then(data => setUsers(data));
  }, []);

  return (
    <div>
```

```
      <h2>User List</h2>
      <ul>
        {users.map(user => <li key={user.id}>{user.name}</li>)}
      </ul>
    </div>
  );
}
export default Users;
```

Outcome: Students learn how to perform asynchronous data fetching and render results dynamically.

PU4.7. Mini Project – Personal Portfolio using React
Objective: Apply all learned concepts to create a functional multi-page web application.
Tasks:

- Create components: Home, About, Projects, Contact.
- Use React Router for navigation.
- Implement dynamic project cards using props and state.
- Style the site with CSS or a framework like Bootstrap.
- Add interactivity with event handling and conditional rendering.

Outcome: Students demonstrate their ability to build a small-scale React project integrating all core concepts.

Module 5: Backend Development with Node.js & Express.js

5.1. Introduction

Modern web applications rely on powerful backend systems to process data, handle requests, and ensure smooth communication between the client and server. The backend acts as the "brain" of a web application — managing databases, authentication, APIs, and business logic that support frontend functionality.

This module introduces students to Node.js and Express.js, two core technologies used in building scalable and efficient server-side applications within the MERN Stack. Node.js allows developers to run JavaScript outside the browser, enabling full-stack JavaScript development using a single language. Express.js, built on top of Node.js, provides a fast, flexible, and minimalist framework for creating web servers and RESTful APIs with ease.

Students will learn how to:

- Set up and configure a Node.js server using Express.js
- Create and manage API routes
- Handle HTTP requests and responses
- Integrate with databases (e.g., MongoDB)
- Implement middleware, routing, and error handling

By mastering these backend fundamentals, learners will be able to build reliable server-side logic, connect their frontends to databases, and develop complete, data-driven web applications.

5.2. Learning Units (LUs)

LU5.2.1. Introduction to Node.js and Express.js

Modern web development requires applications that are fast, scalable, and capable of handling multiple client requests simultaneously. Traditional web servers often struggle with concurrency and real-time data flow, which led to the rise of Node.js, a JavaScript runtime designed for high-performance backend development.

To simplify the process of building web servers and APIs, developers commonly use Express.js, a lightweight and flexible framework built on top of Node.js.

Together, Node.js and Express.js form the foundation of many modern backend systems — including those in the MERN stack enabling developers to build powerful, full-stack JavaScript applications.

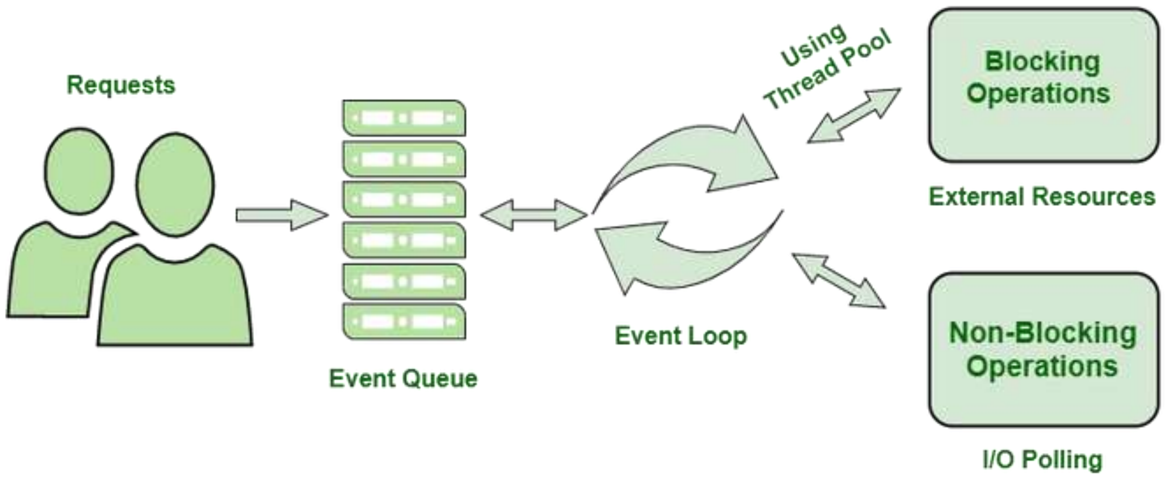
5.2.1.1. What is Node.js?

Node.js is an open-source, cross-platform JavaScript runtime environment built on Google Chrome's V8 engine.

It allows developers to run JavaScript code outside of a web browser primarily on servers making it possible to use JavaScript for backend development.

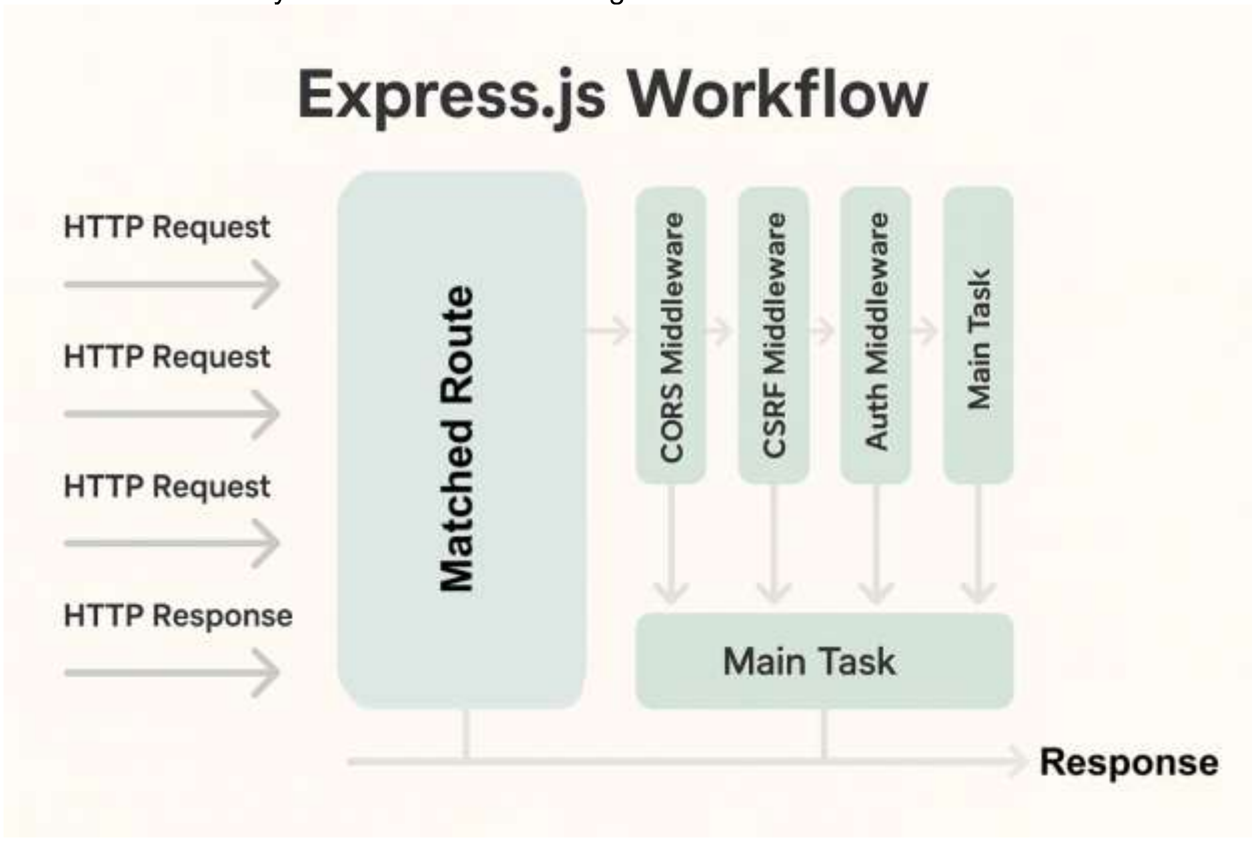
Key Features of Node.js

- **Asynchronous and Event-Driven:**
  Node.js executes multiple requests without blocking, making it ideal for handling real-time data like chats or streaming.
- **Single Programming Language (JavaScript):**
  Enables developers to use JavaScript for both frontend and backend.
- **Fast Execution:**
  Built on the V8 engine, Node.js converts JavaScript into machine code for high-speed performance.
- **Rich Ecosystem:**
  Comes with npm (Node Package Manager), which provides thousands of reusable libraries.
- **Cross-Platform:**
  Works across Windows, macOS, and Linux.



5.2.1.2. What is Express.js?
Express.js is a fast, unopinionated, and minimalist web framework for Node.js that simplifies the process of building web applications and APIs. It provides a set of powerful features for handling HTTP requests, routing, middleware, and error management.
Key Features of Express.js

- **Routing System:**
  Simplifies creating different API endpoints (GET, POST, PUT, DELETE).
- **Middleware Support:**
  Allows inserting custom logic for processing requests (e.g., authentication, logging).
- **Template Engines:**
  Supports rendering dynamic content using EJS, Handlebars, or Pug.
- **Error Handling:**
  Provides centralized error management for cleaner code.
- **Integration Friendly:**
  Works seamlessly with databases like MongoDB.



5.2.1.3. How Node.js and Express.js Work Together

In a typical web application:

- **i.** Node.js runs the server and manages requests from the browser.
- **ii.** Express.js defines routes and handles logic for each request (e.g., /login, /register).
- **iii.** The server processes the request, interacts with the database, and returns the response to the frontend.

Example Code: Basic Express.js Server

```
// Importing Express
const express = require('express');
const app = express();

// Defining a route
app.get('/', (req, res) => {
  res.send('Hello, this is your first Express server!');
});

// Starting the server
app.listen(3000, () => {
  console.log('Server running on http://localhost:3000');
});
```

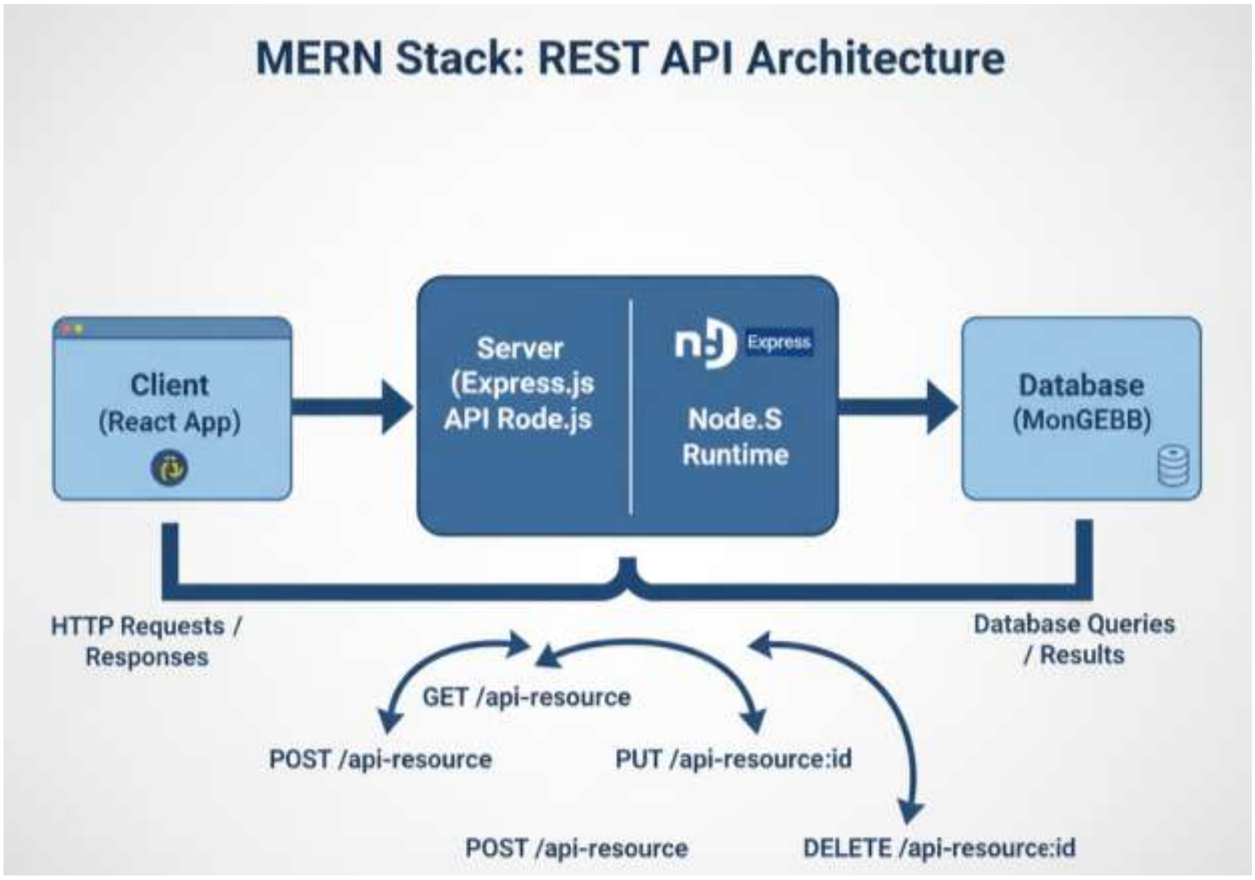
When you run this code (node app.js), visiting
http://localhost:3000 in your browser will display:
“Hello, this is your first Express server!”

LU5.2.2. Setting Up REST APIs and Routes

A REST API (Representational State Transfer Application Programming Interface) is a standardized way for different software systems to communicate over the web using the HTTP protocol. It allows the frontend (client) and backend (server) of a web application to exchange data seamlessly.

In the MERN stack, the React.js frontend sends HTTP requests (like fetching, adding, or deleting data) to the Node.js + Express.js backend. The backend processes these requests, interacts with the MongoDB database if needed, and sends a response back to the client — usually in JSON format.

This unit introduces how to design RESTful APIs, define routes, and handle different types of HTTP methods using Express.js.



5.2.2.1. Understanding RESTful API Principles

REST (Representational State Transfer) is an architectural style for designing networked applications. REST APIs are:

- **Stateless:** Each request from the client contains all necessary information, and the server doesn't store client session data.
- **Resource-Based:** Everything (like users, products, or posts) is treated as a resource, identified by a unique URL.
- **Use of HTTP Methods:**
    - **GET:** Retrieve existing data
    - **POST:** Add new data
    - **PUT:** Update existing data
    - **DELETE:** Remove data
- **JSON Format:** Data is sent and received in lightweight JSON format for easy integration with JavaScript-based frontends.

Example of RESTful Endpoints:

Resource	HTTP Method	Endpoint	Action
Users	GET	/api/users	Get all users
Users	POST	/api/users	Add a new user
Users	PUT	/api/users/:id	Update user by ID
Users	DELETE	/api/users/:id	Delete user by ID

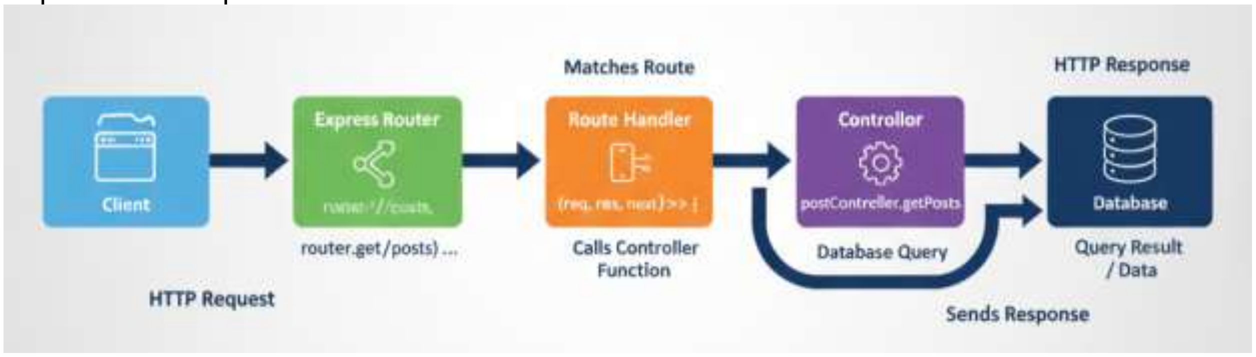
5.2.2.2. Setting Up Express.js Routes

In Express.js, a *route* defines how your server responds to a client request for a particular endpoint and HTTP method.

Example Code:

```
// Import required modules
const express = require('express');
const app = express();

// Middleware to parse JSON data
app.use(express.json());

// Define basic REST API routes
app.get('/api/users', (req, res) => {
  res.send('Get all users');
});

app.post('/api/users', (req, res) => {
  res.send('Add a new user');
});

app.put('/api/users/:id', (req, res) => {
  res.send(`Update user with ID: ${req.params.id}`);
});

app.delete('/api/users/:id', (req, res) => {
  res.send(`Delete user with ID: ${req.params.id}`);
});

// Start the server
app.listen(3000, () => {
  console.log('Server running on port 3000');
});
```

This basic setup forms the backbone of your backend — it defines how your app handles requests and responds with data.



5.2.2.3. Organizing Routes for Scalability

As the project grows, managing all routes in a single file becomes difficult. Therefore, developers separate routes into modules using the Express Router.

Example (User Routes in a separate file):

```
// userRoutes.js
const express = require('express');
const router = express.Router();

// Example routes
router.get('/', (req, res) => res.send('Get all users'));
router.post('/', (req, res) => res.send('Add a new user'));
router.put('/:id', (req, res) => res.send(`Update user ${req.params.id}`));
router.delete('/:id', (req, res) => res.send(`Delete user ${req.params.id}`));

module.exports = router;
```

And in server.js**:**

```
const express = require('express');
const app = express();
const userRoutes = require('./routes/userRoutes');

app.use('/api/users', userRoutes);

app.listen(3000, () => console.log('Server running on port 3000'));
```

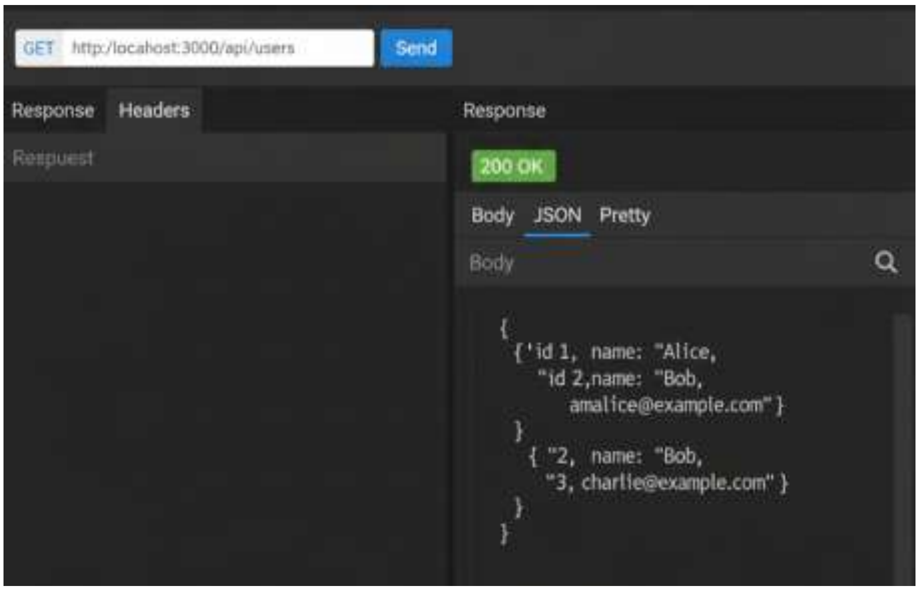
This modular approach improves readability, reusability, and maintainability of your code.

5.2.2.4. Testing APIs Using Postman

Once the routes are created, they should be tested using Postman or a similar API testing tool.

- **Step 1:** Open Postman
- **Step 2:** Enter URL like http://localhost:3000/api/users
- **Step 3:** Choose request type (GET, POST, PUT, DELETE)
- **Step 4:** Click Send
- **Step 5:** Observe the response message from the server

This helps verify that the API endpoints are working as expected before integrating them with the frontend.



```
{
  {'id 1,  name: "Alice,
     "id 2,name: "Bob,
        amalice@example.com" }
  }
   { "2,  name: "Bob,
     "3, charlie@example.com" }
  }
}
```

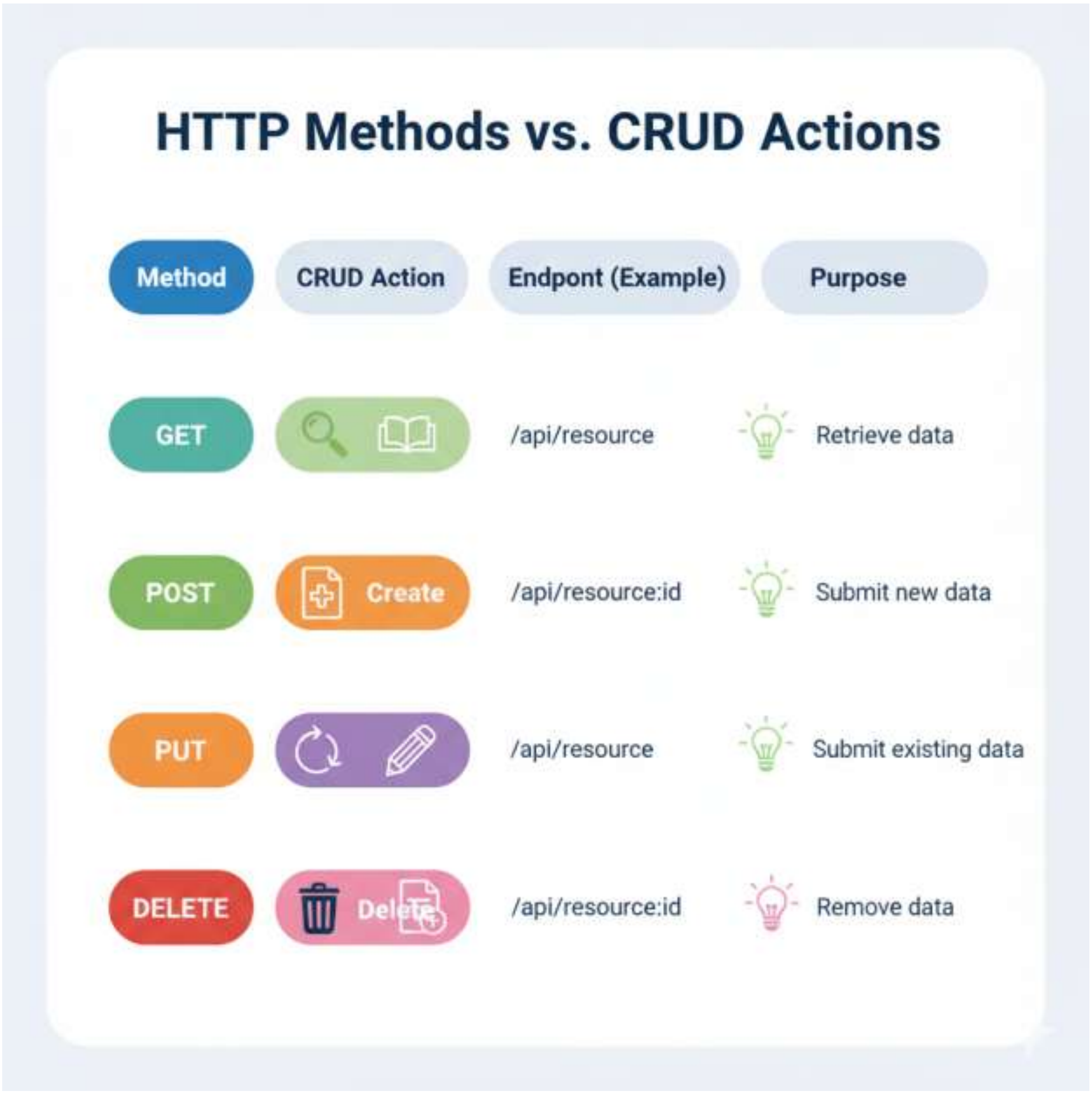
LU5.2.3. Working with HTTP Methods (GET, POST, PUT, DELETE)

Every web application needs to communicate between the client (frontend) and server (backend). This communication happens using the HTTP protocol, where the client sends requests, and the server responds.

The HTTP methods — GET, POST, PUT, and DELETE — define the type of operation being performed on the server. In RESTful API design, these methods correspond to the basic CRUD operations:

HTTP Method	CRUD Operation	Description
GET	Read	Retrieve data from the server
POST	Create	Send new data to the server
PUT	Update	Modify existing data on the server
DELETE	Delete	Remove data from the server

Understanding and correctly implementing these methods in Express.js is fundamental for building functional and well-structured backend APIs.



HTTP Methods vs. CRUD Actions

Method	CRUD Action	Endpont (Example)	Purpose
GET		/api/resource	Retrieve data
POST	Create	/api/resource:id	Submit new data
PUT		/api/resource	Submit existing data
DELETE	Delete	/api/resource:id	Remove data

5.2.3.1. GET Method (Retrieve Data)

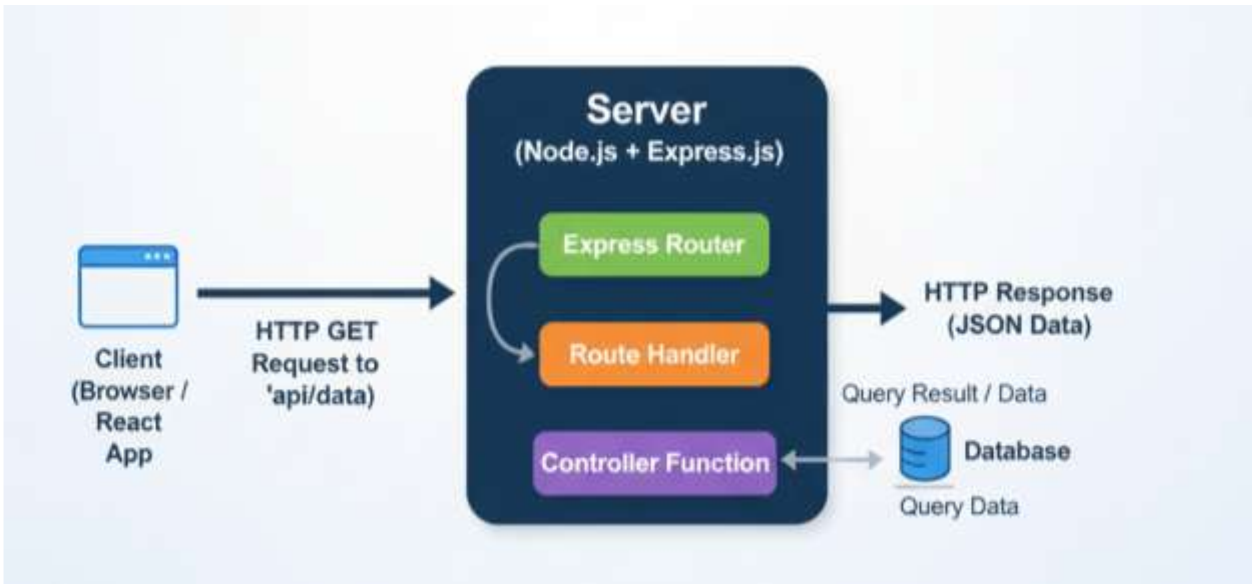
The GET method is used to retrieve data from the server. It does not modify any data it simply fetches and returns it.

Example:

```
app.get('/api/products', (req, res) => {
  const products = [
    { id: 1, name: 'Laptop', price: 900 },
    { id: 2, name: 'Keyboard', price: 50 },
  ];
  res.json(products);
});
```

Explanation:

- app.get() defines a route for retrieving data.
- When a client sends a GET request to /api/products, the server responds with a list of products in JSON format.



5.2.3.2. POST Method (Create Data)

The POST method is used to send data to the server to create a new resource. It typically includes a request body containing the data to be added.

Example:

```
app.post('/api/products', (req, res) => {
  const newProduct = req.body;
  res.status(201).json({
    message: 'Product created successfully',
    data: newProduct,
  });
});
```

Explanation:

- app.post() defines a route to create new entries.
- req.body contains the data sent by the client (like a new product).
- res.status(201) indicates successful creation.

5.2.3.3. PUT Method (Update Data)

The PUT method is used to update or modify existing data on the server. It usually includes the resource ID in the URL.

Example:

```
app.put('/api/products/:id', (req, res) => {
  const { id } = req.params;
  const updatedProduct = req.body;
  res.json({
    message: `Product with ID ${id} updated successfully`,
    updatedData: updatedProduct,
  });
});
```

Explanation:

- :id is a route parameter representing the product ID.
- The server uses this ID to locate the product and update its data.

5.2.3.4. DELETE Method (Remove Data)

The DELETE method is used to remove a specific resource from the server, usually identified by its ID.

Example:

```
app.delete('/api/products/:id', (req, res) => {
  const { id } = req.params;
  res.json({
    message: `Product with ID ${id} deleted successfully`,
  });
});
```

Explanation:

- The client specifies the resource to delete through the URL.
- The server confirms the deletion with a response message.

5.2.3.5. Using Postman to Test HTTP Methods

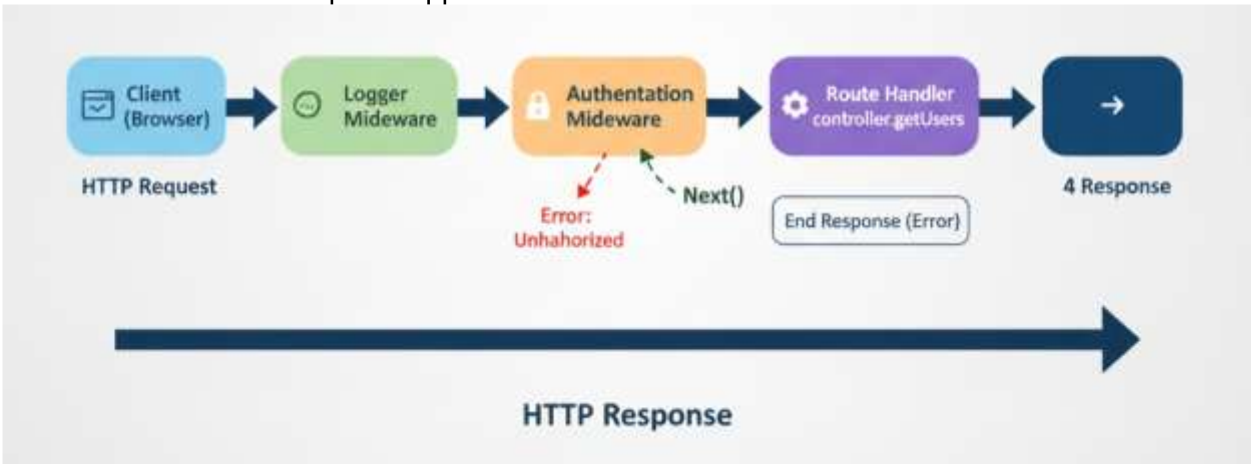
Each method can be tested using Postman:

i. **GET** → Retrieve existing data
ii. **POST** → Add new item (include JSON body in "Body → raw → JSON")
iii. **PUT** → Update an existing item by ID
iv. **DELETE** → Remove item by ID

Testing these endpoints ensures that your routes are functioning correctly before connecting them to a frontend like React.

LU5.2.4. Understanding Middleware in Express

In Express.js, middleware plays a central role in handling requests and responses. Middleware functions act as intermediaries that sit between the client request and the server's final response. They can inspect, modify, validate, or log data as it passes through the application. In simple terms, middleware is a function that has access to the request (req), response (res), and the next middleware function in the application's request-response cycle.
This unit introduces learners to what middleware is, how it works, and how to create and use custom middleware in Express applications.



5.2.4.1. What is Middleware?

Middleware functions in Express are functions that execute during the lifecycle of a request to the server. Each middleware has access to:

- The request object (req)
- The response object (res)
- The next function (next) that passes control to the next middleware in the stack

These functions can:

- Execute any code
- Modify req and res objects
- End the request-response cycle
- Call the next() function to move to the next middleware

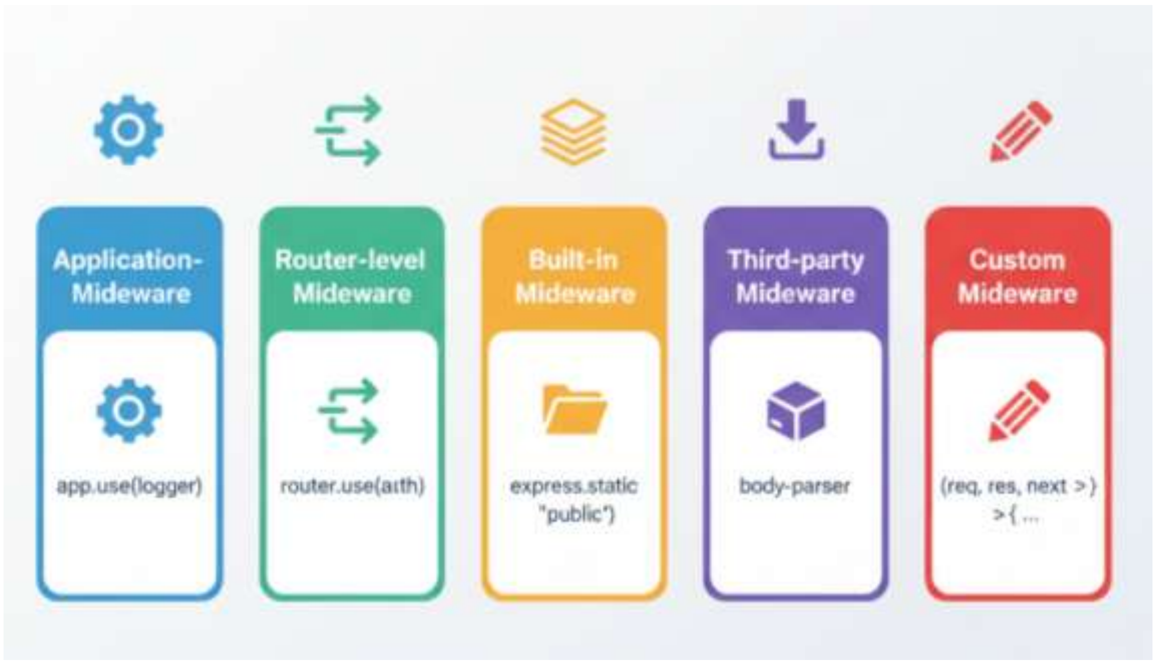
Syntax Example:

```
app.use((req, res, next) => {
  console.log('Middleware executed');
  next(); // Pass control to the next middleware or route handler
});
```

5.2.4.2. Types of Middleware in Express

There are several types of middleware, each serving a specific purpose:

Type	Description	Example
Application-level	Bound to an Express app instance using app.use()	Logging, authentication
Router-level	Bound to specific route paths	/users, /products
Built-in	Provided by Express itself	express.json(), express.static()
Third-party	Installed via npm packages	morgan, cors, helmet
Custom	Created by developers	Validation, error handling



5.2.4.3. Built-in Middleware Examples

Express comes with a few built-in middleware functions that handle common tasks.

a. JSON Parsing:

```
app.use(express.json());
```

→ Parses incoming JSON data in requests (e.g., from POST or PUT).

b. Static Files:

```
app.use(express.static('public'));
```

→ Serves static assets (HTML, CSS, images, JS) from a folder.

5.2.4.4. Third-Party Middleware Examples

You can use community-built middleware to enhance your app.

Example: Using morgan for request logging

```
const morgan = require('morgan');
app.use(morgan('dev'));
```

→ Automatically logs request details (method, path, status code).

Example: Using cors for cross-origin access

```
const cors = require('cors');
app.use(cors());
```

→ Enables your frontend (React) to access backend APIs securely.

5.2.4.5. Creating Custom Middleware

Developers can also create custom middleware to handle specific logic — like authentication or request validation.

Example:

```
const checkAuth = (req, res, next) => {
  if (req.headers.authorization === 'secret-token') {
    next(); // Proceed if authorized
  } else {
    res.status(403).json({ message: 'Access denied' });
  }
};

app.use(checkAuth);
```

Explanation:

- Checks if the incoming request contains a valid authorization token.
- If valid → proceeds to next middleware.
- If invalid → returns a "403 Forbidden" error.

5.2.4.6. Middleware Execution Flow

When multiple middleware are used, they execute in the order they are defined.

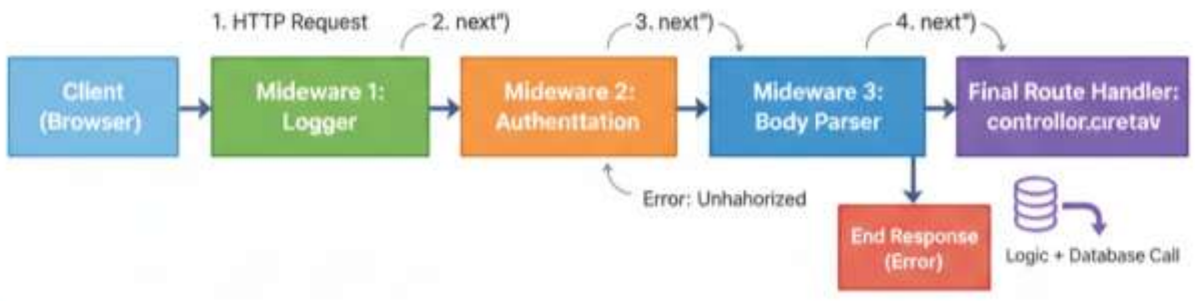
Example:

```
app.use((req, res, next) => {
  console.log('First middleware');
  next();
});

app.use((req, res, next) => {
  console.log('Second middleware');
  next();
});

app.get('/', (req, res) => {
  res.send('Hello from the final route!');
});
```

Output sequence (in console):

```
First middleware
Second middleware
```

Middleware functions run in sequence, before the final route sends a response.

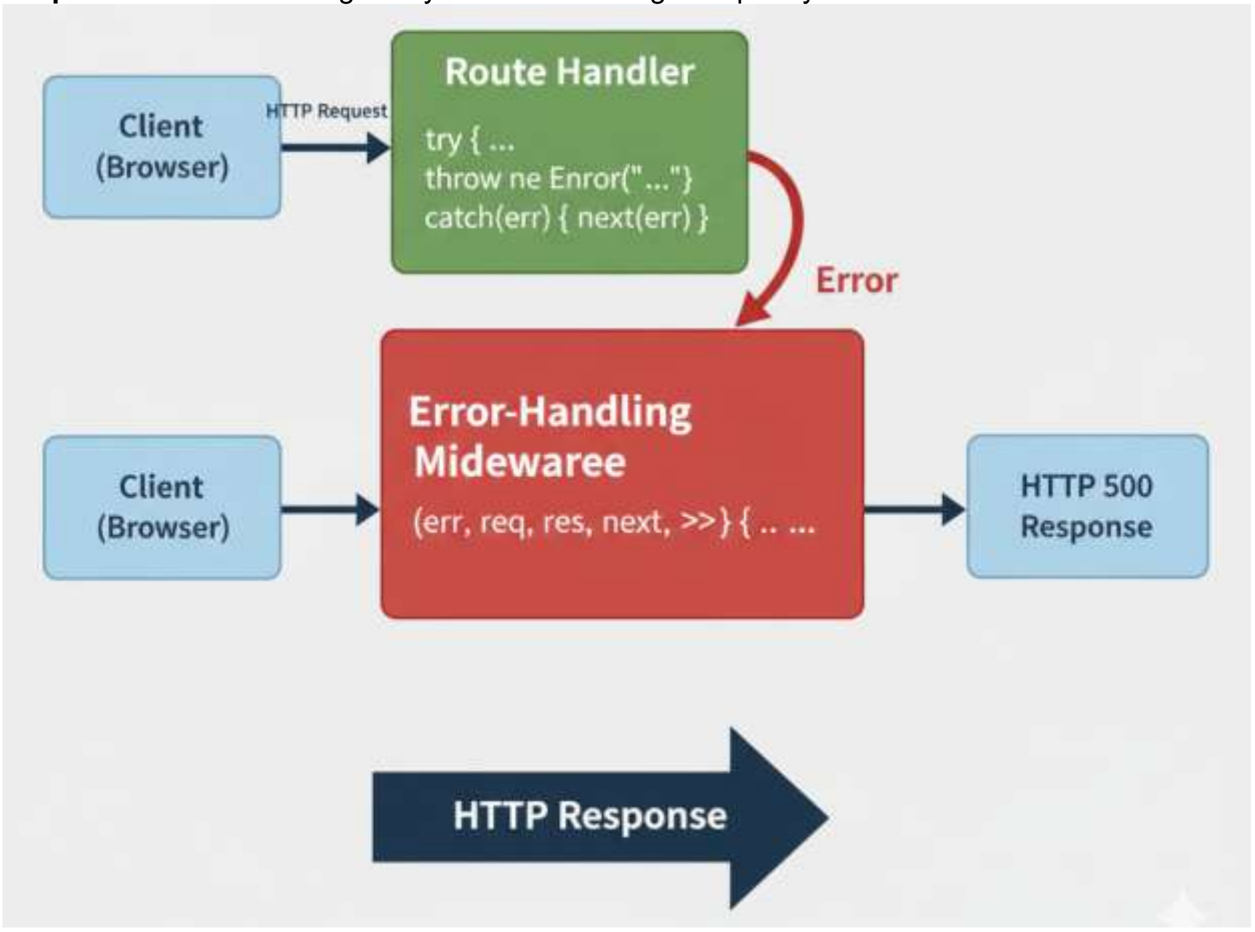


5.2.4.7. Error-Handling Middleware

Express provides special error-handling middleware — functions with four parameters (err, req, res, next).

Example:

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});
```
Purpose: Catches errors globally instead of writing multiple try-catch blocks in each route.



5.2.4.8. Real-Life Example: Middleware Chain

Imagine a request flow in your Express app:

- express.json() parses the request body.
- cors() allows frontend access.
- checkAuth() verifies login.
- Route handler processes data.
- Error handler catches any exceptions.

This creates a pipeline of functions, ensuring your app processes every request securely and efficiently.

5.3. Practical Units (PUs)

PU5.3.1. Setting Up Node.js and Express Server

Objective: To create and run the first backend server using Node.js and Express.

Tasks:

- Initialize a new Node.js project using npm init.
- Install Express using npm install express.
- Create a basic Express server (server.js) that responds with “Hello World!” when accessed through a browser or Postman.
- Run the server using node server.js and verify the response on localhost.

Expected Outcome: Students will understand how to set up a Node.js environment and launch an Express-based web server.

PU5.3.2. Creating RESTful API Routes
Objective: To build basic routes for CRUD operations using Express routing.
Tasks:

- Create a new Express app.
- Define routes for /api/users to handle GET (fetch all users) and POST (add new user) requests.
- Use Express Router for modular route management.
- Test all routes using Postman.

Expected Outcome: Students will be able to define and manage multiple API routes in Express.

PU5.3.3. Handling HTTP Methods (GET, POST, PUT, DELETE)
Objective: To implement and test various HTTP methods for CRUD functionality.
Tasks:

- Create routes for:
    - **GET** /api/products – Retrieve all products
    - **POST** /api/products – Add a new product
    - **PUT** /api/products/:id – Update an existing product
    - **DELETE** /api/products/:id – Delete a product
- Use dummy JSON data for product storage (no database yet).
- Test each endpoint in Postman.

Expected Outcome: Students will understand CRUD operations and API endpoint testing using HTTP methods.

PU5.3.4. Using Middleware in Express
Objective: To explore built-in, third-party, and custom middleware in Express.
Tasks:

- Implement a simple logging middleware to display request details (method, URL, and time).
- Use body-parser middleware to handle JSON data.
- Add error-handling middleware for handling invalid routes or server errors.
- Demonstrate middleware chaining and execution order.

Expected Outcome: Students will understand the purpose of middleware and how to use it for request handling and error management.

PU5.3.5. Building a Mini REST API Project
Objective: To integrate all learned concepts into a functional backend application.
Tasks:

- Create a mini API for a “Student Management System”.
- Implement routes for managing student records (add, update, view, delete).
- Apply middleware for validation and error handling.
- Use Postman to test all routes.

Expected Outcome: Students will gain hands-on experience in developing a small-scale RESTful API backend using Node.js and Express.js.

Module 6: Database Integration with MongoDB & Deployment

6.1 Introduction

This module introduces students to database integration using MongoDB, one of the most widely used NoSQL databases in modern web development. Learners will explore how to connect Node.js and Express.js applications to MongoDB, perform CRUD operations, and manage data efficiently.
The module also covers deployment techniques for hosting MERN applications on cloud platforms such as Render, Vercel, or MongoDB Atlas, preparing learners for real-world web app deployment.
By the end of this module, learners will be able to:

- Set up and connect MongoDB with Node.js using Mongoose.
- Design and manage data models for backend systems.
- Perform CRUD operations on MongoDB collections.
- Deploy a full-stack MERN application to the cloud.

6.2 Learning Units (LUs)

LU6.2.1. Introduction to NoSQL and MongoDB

Databases are a core part of any web application, enabling the storage, retrieval, and management of data. Traditionally, developers used Relational Databases (RDBMS) such as MySQL or PostgreSQL, which store data in structured tables with predefined schemas. However, modern web applications require handling large volumes of unstructured or semi-structured data, which led to the rise of NoSQL (Not Only SQL) databases.
MongoDB is one of the most popular NoSQL databases, designed for scalability, flexibility, and performance. It stores data in a document-oriented format, making it ideal for use in the MERN stack (MongoDB, Express.js, React.js, Node.js).

6.2.1.1. Understanding NoSQL Databases

NoSQL refers to a category of databases that use non-tabular data models — unlike traditional SQL databases.
Instead of rows and columns, NoSQL databases store data as documents, key-value pairs, graphs, or wide-column stores.

Types of NoSQL Databases:

i. **Document-Based** – e.g., MongoDB, CouchDB
→ Store data as JSON-like documents.
ii. **Key-Value Stores** – e.g., Redis, DynamoDB
→ Use a simple key-value pair structure.
iii. **Column-Based** – e.g., Cassandra, HBase
→ Store data in columns for analytical workloads.
iv. **Graph-Based** – e.g., Neo4j
→ Focus on relationships between data entities.

Key Advantages of NoSQL:

- Flexible and schema-less data model
- Scales horizontally (easy to handle large data)
- Fast read/write performance
- Ideal for real-time web apps

6.2.1.2. Introduction to MongoDB

MongoDB is a document-oriented NoSQL database that stores data in BSON (Binary JSON) format.
It allows developers to represent data in the same JSON format used by JavaScript applications, making integration seamless in MERN stack development.

Basic Concepts in MongoDB:

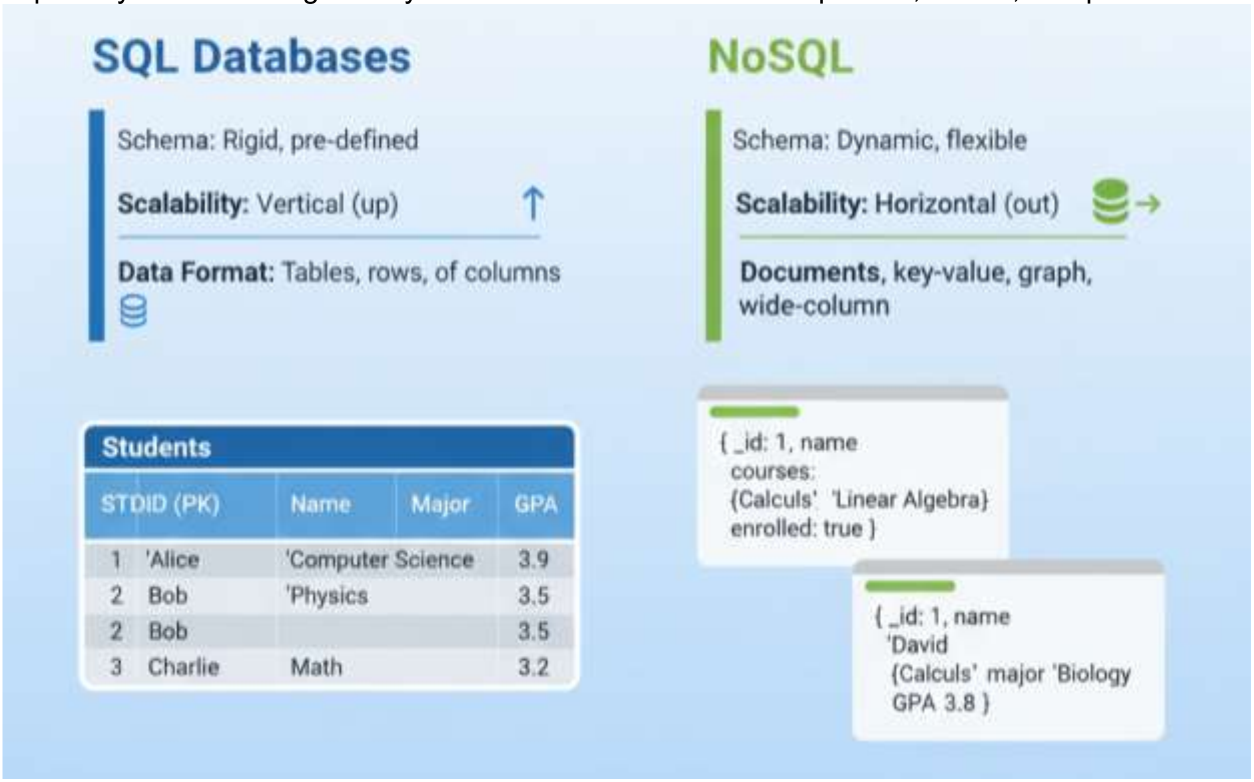
Concept	Description	Example
Database	A container for collections.	schoolDB
Collection	Similar to a table; holds multiple documents.	students
Document	A single record in JSON format.	{ "name": "Ali", "age": 20 }
Field	A key-value pair inside a document.	"name": "Ali"
BSON	Binary JSON used for data storage and retrieval.	Efficient for performance

MongoDB stores data in a flexible structure, allowing documents in the same collection to have different fields — unlike rigid SQL tables.

6.2.1.3. Comparing SQL vs NoSQL (MongoDB)

Feature	SQL Database	NoSQL (MongoDB)
Structure	Tables with rows and columns	Collections with documents
Schema	Fixed	Dynamic/Flexible
Query Language	SQL	MongoDB Query Language (MQL)
Scalability	Vertical (add power to one machine)	Horizontal (add more servers)
Joins	Supported	Not directly (use embedding or referencing)
Data Format	Tabular	JSON/BSON

MongoDB's document-based approach simplifies how developers store real-world data — especially when working with dynamic or nested data like user profiles, orders, and posts.


SQL Databases
Schema: Rigid, pre-defined
Scalability: Vertical (up)
Data Format: Tables, rows, of columns
Students
STDID (PK)
Name
Major
GPA
1 'Alice 'Computer Science 3.9
2 Bob 'Physics 3.5
2 Bob 3.5
3 Charlie Math 3.2
NoSQL
Schema: Dynamic, flexible
Scalability: Horizontal (out)
Documents, key-value, graph, wide-column
{ _id: 1, name courses: {Calculs' 'Linear Algebra} enrolled: true }
{ _id: 1, name 'David {Calculs' major 'Biology GPA 3.8 }

6.2.1.4. Installing MongoDB

MongoDB can be installed locally or used through MongoDB Atlas (Cloud).

Option 1: Local Installation

i. Go to https://www.mongodb.com/try/download/community
ii. Download and install MongoDB Community Server.
iii. Verify installation using:

iv. mongod --version
v. mongo --version

Option 2: MongoDB Atlas (Recommended)

i. Visit https://www.mongodb.com/cloud/atlas
ii. Create a free cluster.
iii. Get your connection URI (example):
iv. mongodb+srv://<username>:<password>@cluster0.mongodb.net/myDatabase
v. Use this URI in your Node.js app for connection.

6.2.1.5. Real-World Example

Example Document in MongoDB:

```
{
  "studentID": 101,
  "name": "Ayesha Khan",
  "email": "ayesha@example.com",
  "courses": ["Web Development", "AI Fundamentals"],
  "isActive": true
}
```

This structure is flexible you can add or remove fields anytime without altering the entire database schema.

LU6.2.2. Connecting MongoDB with Express.js

Once MongoDB is installed or set up on the cloud (e.g., MongoDB Atlas), the next step in full-stack development is to connect the database with the backend server. In the MERN stack, this is typically done using Express.js a lightweight Node.js framework and Mongoose, a popular ODM (Object Data Modeling) library that provides a simple way to work with MongoDB data.

This learning unit focuses on establishing a connection between MongoDB and Express.js, defining a database schema, and performing basic data operations.

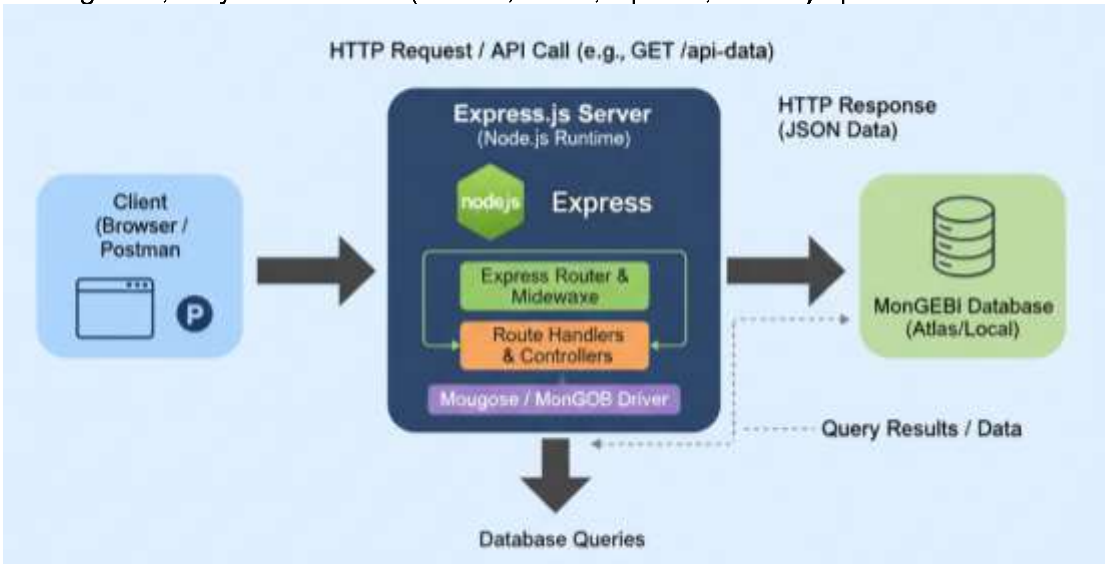
6.2.2.1. Understanding the Role of Express.js and MongoDB Connection

In a web application, Express.js acts as the backend framework that manages routes, middleware, and APIs.

MongoDB, on the other hand, serves as the database that stores user, product, or other application data.

To connect both:

- Express.js handles the incoming HTTP requests (like /api/users)
- Mongoose connects the Express server to the MongoDB database
- Together, they allow CRUD (Create, Read, Update, Delete**)** operations



6.2.2.2. Installing Required Packages

Before writing the code, ensure Node.js and npm are properly installed. Create a new project folder and initialize it using:

```
npm init -y
```

Then install the required packages:

```
npm install express mongoose
```

Here:

- express → used to build RESTful APIs
- mongoose → used to interact with MongoDB database

6.2.2.3. Setting Up a Basic Express Server

Create a file named server.js in your project folder and add the following code:

```
const express = require('express');
const mongoose = require('mongoose');

const app = express();
const PORT = 5000;

// Middleware
app.use(express.json());

// Basic route
app.get('/', (req, res) => {
  res.send('Express & MongoDB connection example');
});

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

This code initializes a simple Express server that listens on port 5000.
Run it using:

```
node server.js
```

Check in the browser or Postman by visiting:
http://localhost:5000
You should see: “Express & MongoDB connection example”

6.2.2.4. Connecting Express to MongoDB

Now, let’s connect this server to MongoDB.
You can use either MongoDB Atlas (Cloud) or a local MongoDB instance.
Example connection code (add below middleware in server.js):

```
mongoose.connect('mongodb://127.0.0.1:27017/mydatabase', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
})
.then(() => console.log('✅ MongoDB Connected Successfully'))
.catch((err) => console.log('❌ MongoDB Connection Error:', err));
```

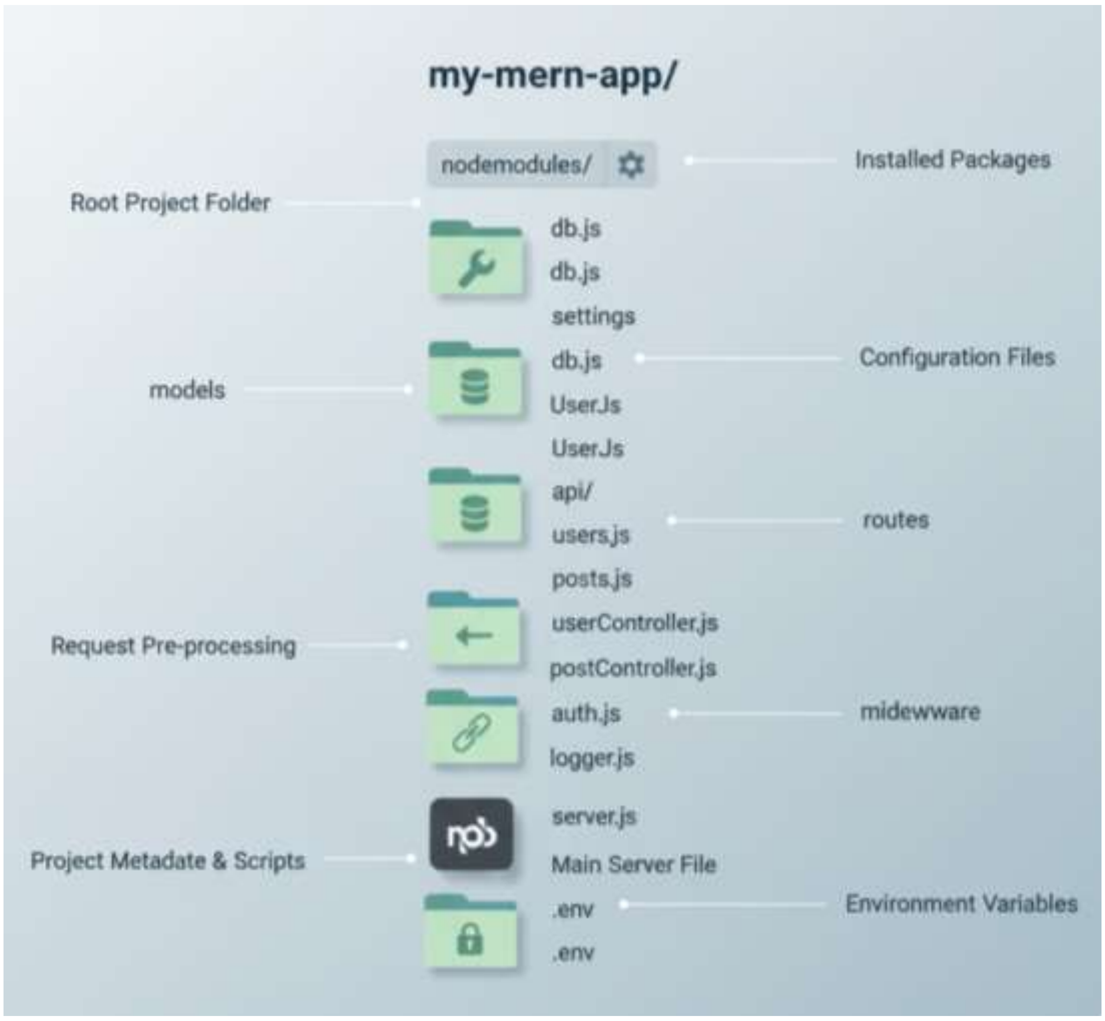
If you’re using MongoDB Atlas, replace the URI with your cluster connection string:

```
mongoose.connect('mongodb+srv://<username>:<password>@cluster0.mongodb.net/myDatabase')
```

Once connected, your terminal should display:
MongoDB Connected Successfully

6.2.2.5. Defining a MongoDB Schema and Model

A Schema defines the structure of documents stored in a collection.
A Model is a compiled version of that schema that interacts directly with the database.



Example: Creating a User model in a new file named models/User.js

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: Number,
});
module.exports = mongoose.model('User', userSchema);
```

This schema defines that each user document will have:

- name (String, required)
- email (String, required, unique)
- age (Number)

6.2.2.6. Performing Basic CRUD Operations

To test the database connection and model, create a sample route in server.js:

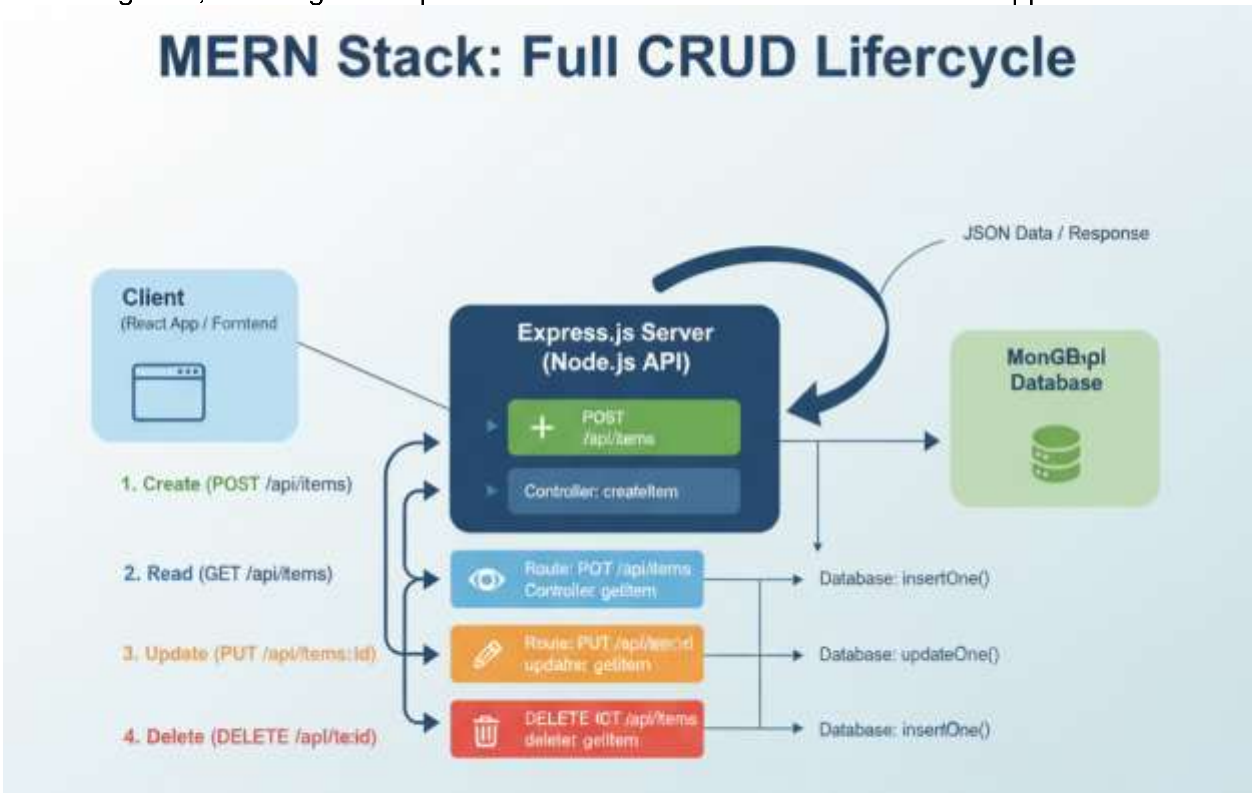
```
const User = require('./models/User');
// POST route - Add a new user
app.post('/api/users', async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.status(201).send(user);
  } catch (err) {
    res.status(400).send(err);
  }
});
```

```
// GET route - Retrieve all users
app.get('/api/users', async (req, res) => {
  try {
    const users = await User.find();
    res.send(users);
  } catch (err) {
    res.status(500).send(err);
  }
});
```
Test these routes using Postman or Thunder Client (VS Code Extension). When you add a user via POST request, it will be stored in MongoDB, confirming your connection works.

LU6.2.3. Performing CRUD Operations (Create, Read, Update, Delete)
After connecting MongoDB with Express.js, the next essential step is performing CRUD operations — the foundation of backend data handling.
CRUD stands for:

- **C** – *Create* new data
- **R** – *Read* existing data
- **U** – *Update* existing data
- **D** – *Delete* unwanted data

In a typical MERN stack application, CRUD operations allow interaction between the user interface (frontend) and the database (backend).
This Learning Unit explains how to perform CRUD operations using Express.js, MongoDB, and Mongoose, enabling developers to build functional and data-driven web applications.



6.2.3.1. Understanding CRUD Operations in Context
CRUD forms the basis of any data management system.
When a user submits a form, views records, edits information, or deletes entries, CRUD operations are being executed behind the scenes.

Operation	HTTP Method	Example Route	Description
Create	POST	/api/users	Adds a new user to the database
Read	GET	/api/users	Retrieves one or more users
Update	PUT	/api/users/:id	Updates user details by ID
Delete	DELETE	/api/users/:id	Removes a user from the database

6.2.3.2. Creating a Model

Before performing operations, define a Mongoose model.

Example: models/User.js

```
const mongoose = require('mongoose');
const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique: true },
  age: Number
});
module.exports = mongoose.model('User', userSchema);
```

This schema defines how the “users” collection in MongoDB will store documents.

6.2.3.3. Setting Up CRUD Routes in Express

In your main backend file (server.js), import dependencies and the model:

```
const express = require('express');
const mongoose = require('mongoose');
const User = require('./models/User');
const app = express();
app.use(express.json());
```

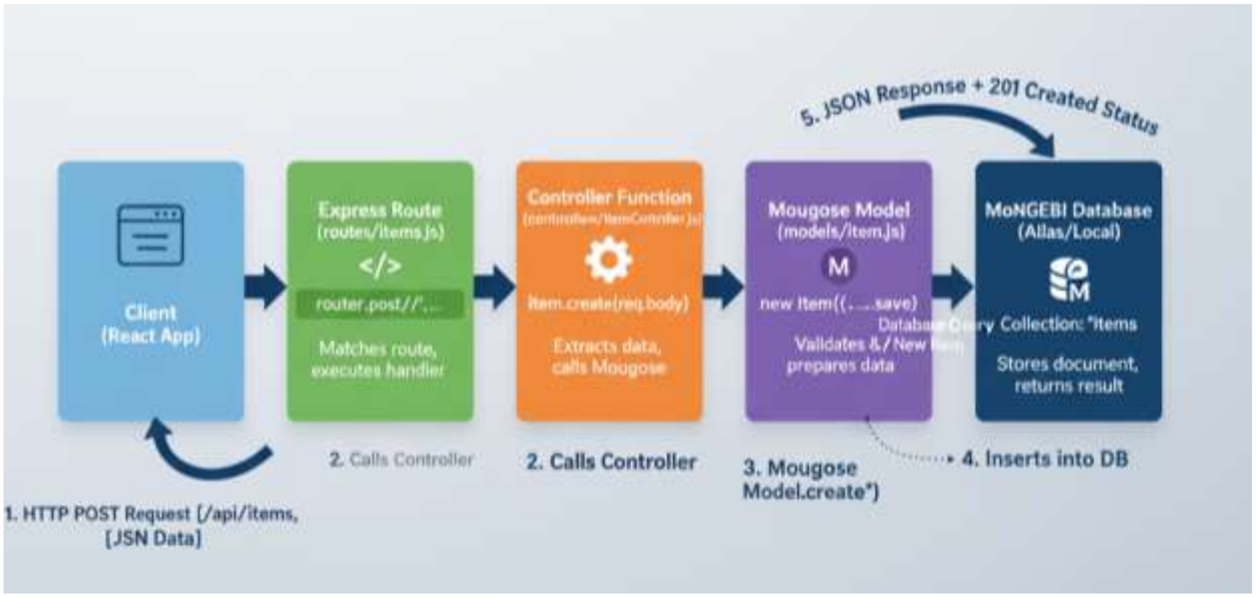
6.2.3.4. CREATE (POST Request)

Used to add new data to the MongoDB collection.

```
app.post('/api/users', async (req, res) => {
  try {
    const newUser = new User(req.body);
    await newUser.save();
    res.status(201).send(newUser);
  } catch (err) {
    res.status(400).send(err);
  }
});
```

Explanation:

This route accepts JSON data (e.g., { "name": "Ali", "email": "ali@example.com", "age": 25 }) and stores it in the MongoDB collection.


5. JSON Response + 201 Created Status
Client (React App)
Express Route (routes/items.js)
</>
router.post('/',...
Matches route, executes handler
Controller Function (controllers/itemController.js)
Item.create(req.body)
Extracts data, calls Mougose
Mougose Model (models/item.js)
M
new Item((.....save)
Validates &/ New
prepares data
MoNGEBI Database (Atlas/Local)
Database Query Collection: "items
Stores document, returns result
1. HTTP POST Request [/api/items, [JSN Data]
2. Calls Controller
2. Calls Controller
3. Mougose Model.create")
4. Inserts into DB

6.2.3.5. READ (GET Request)

Used to retrieve data from MongoDB.

```
app.get('/api/users', async (req, res) => {
  try {
    const users = await User.find();
    res.send(users);
  } catch (err) {
    res.status(500).send(err);
  }
});
```

Explanation:

This route fetches all user documents and returns them as an array of JSON objects.

You can also fetch a single user by ID:

```
app.get('/api/users/:id', async (req, res) => {
  try {
    const user = await User.findById(req.params.id);
    res.send(user);
  } catch (err) {
    res.status(404).send({ message: 'User not found' });
  }
});
```

6.2.3.6. UPDATE (PUT Request)

Used to modify existing data.

```
app.put('/api/users/:id', async (req, res) => {
  try {
    const updatedUser = await User.findByIdAndUpdate(
      req.params.id,
      req.body,
      { new: true }
    );
    res.send(updatedUser);
  } catch (err) {
    res.status(400).send(err);
  }
});
```

Explanation:

The findByIdAndUpdate() function locates a record using its ID and updates it with the

provided data.
The { new: true } option ensures the updated document is returned in the response.

6.2.3.7. DELETE (DELETE Request)

Used to remove data from MongoDB.

```
app.delete('/api/users/:id', async (req, res) => {
  try {
    await User.findByIdAndDelete(req.params.id);
    res.send({ message: 'User deleted successfully' });
  } catch (err) {
    res.status(500).send(err);
  }
});
```

Explanation:

This route deletes a document by its unique MongoDB _id.

6.2.3.8. Testing CRUD Operations

Use Postman or Thunder Client in VS Code to test your endpoints:

- **POST** → Add new users
- **GET** → View users
- **PUT** → Update existing user
- **DELETE** → Remove a user

LU6.2.4. Introduction to Mongoose for Data Modeling

6.2.4.1. Introduction

When building Node.js and Express.js applications, you often need to interact with a MongoDB database to store, retrieve, and manage data.
However, MongoDB is schema-less — meaning it does not enforce a fixed structure for documents. While flexible, this can lead to inconsistencies if not handled carefully.
This is where Mongoose comes in.
Mongoose is an Object Data Modeling (ODM) library for MongoDB and Node.js that provides a straightforward way to:

- Define data schemas (structure and validation),
- Perform CRUD operations easily,
- Manage relationships between data, and
- Handle data validation and middleware logic.

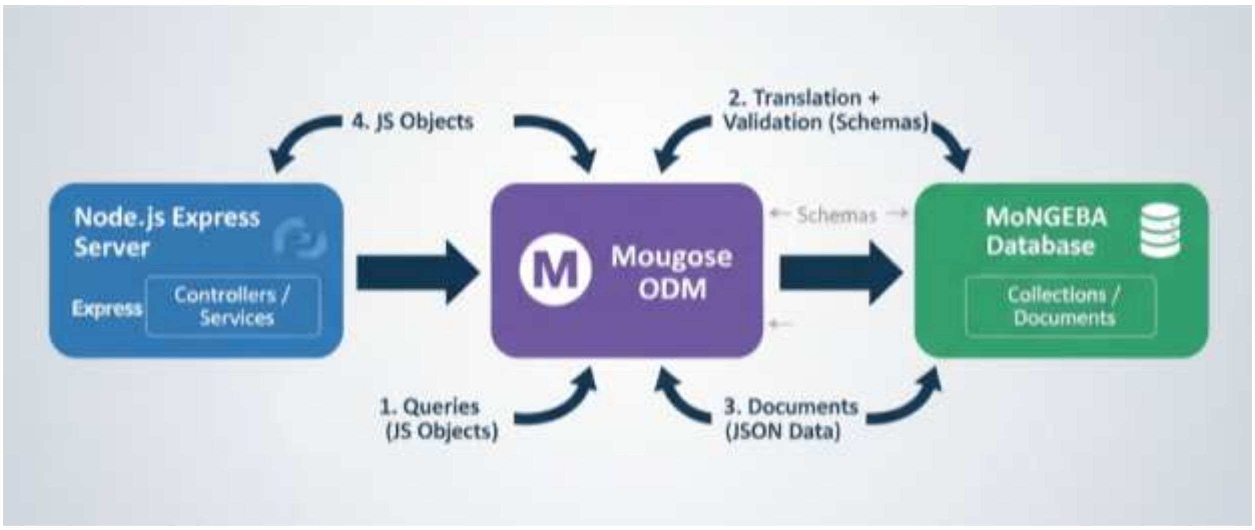
In short, Mongoose acts as a bridge between your Express.js application and the MongoDB database.

6.2.4.2. What is Mongoose?

Mongoose simplifies the interaction between the Node.js application and MongoDB by providing a schema-based structure to the data.

Key Features:

- Schema-based modeling for defining document structure.
- Built-in validation for fields (e.g., required, unique, min/max).
- Easy-to-use query functions (e.g., find(), save(), updateOne()).
- Middleware hooks for logic before/after saving or updating documents.
- Virtual fields and methods for data transformation.



6.2.4.3. Installing and Importing Mongoose

Install Mongoose in your Node.js project using npm:

```
npm install mongoose
```

Then, import and connect it to your MongoDB database:

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/studentDB')
  .then(() => console.log('MongoDB connected successfully!'))
  .catch((err) => console.error('Database connection error:', err));
```

Explanation:

This code connects the backend application to a local MongoDB database named studentDB. You can replace the URL with your MongoDB Atlas connection string for cloud-based databases.

6.2.4.4. Defining a Schema

Schemas define the structure of your MongoDB documents, similar to a table definition in SQL databases.

Example Student Schema:

```
const studentSchema = new mongoose.Schema({
  name: { type: String, required: true },
  rollNo: { type: Number, unique: true },
  email: { type: String, required: true },
  department: String,
  cgpa: { type: Number, min: 0, max: 4.0 }
});
```

Explanation:

Each field has a data type and optional constraints such as required, unique, or min/max values.

This ensures that data stored in MongoDB remains consistent and valid.

6.2.4.5. Creating a Model

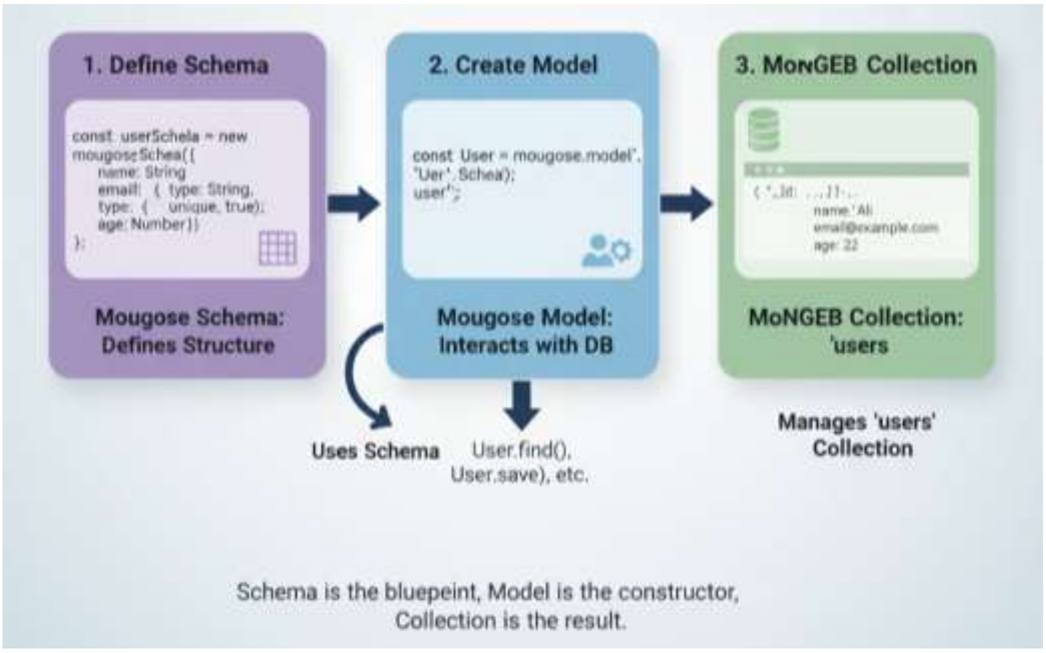
A model is a compiled version of a schema.

It provides methods to interact with the database, such as saving, finding, or deleting documents.

Explanation:

Here, Student represents the collection students in the MongoDB database.

Through this model, we can perform all CRUD operations.


1. Define Schema
2. Create Model
3. MoNGEB Collection
Mougose Schema: Defines Structure
Mougose Model: Interacts with DB
MoNGEB Collection: 'users
Uses Schema
Manages 'users' Collection
Schema is the bluepeint, Model is the constructor, Collection is the result.

```
const userSchela = new
mougose.Schea({
    name: String
    email: { type: String,
    type: {    unique, true);
    age: Number})
};
```

```
const User = mougose.model',
'Uer', Schea);
user';
```

```
{ '_Id: ...}}..
        name:'Ali
        email@example.com
        age: 22
```

```
User.find(),
User.save), etc.
```

6.2.4.6. Creating and Saving Data

Example of inserting a new record into the database:

```
const newStudent = new Student({
  name: 'Ali Khan',
  rollNo: 101,
  email: 'ali.khan@example.com',
  department: 'Computer Science',
  cgpa: 3.8
});

newStudent.save()
  .then(() => console.log('Student record added successfully!'))
  .catch((err) => console.log('Error saving data:', err));
```

Explanation:

This command creates a new document and saves it to the MongoDB collection using the Mongoose model.

6.2.4.7. Reading Data

You can easily fetch all documents or query specific ones:

```
Student.find()
  .then(students => console.log(students))
  .catch(err => console.log(err));
```

To find specific data:

```
Student.findOne({ rollNo: 101 })
  .then(student => console.log(student))
  .catch(err => console.log(err));
```

6.2.4.8. Updating Data

To update existing records:

```
Student.updateOne({ rollNo: 101 }, { cgpa: 3.9 })
  .then(() => console.log('Student updated successfully!'))
  .catch(err => console.log(err));
```

6.2.4.9. Deleting Data

To delete a record from MongoDB:

```
Student.deleteOne({ rollNo: 101 })
  .then(() => console.log('Student deleted successfully!'))
  .catch(err => console.log(err));
```

6.2.4.10. Advantages of Using Mongoose

Feature	Description
Schema Definition	Enforces structure on MongoDB collections.
Validation	Ensures only valid data is stored.
Query Builder	Provides easy-to-use query functions.
Middleware Support	Allows pre/post hooks for operations.
Relationship Management	Supports references between collections.
Error Handling	Catches and manages database errors efficiently.

6.3 Practical Units (PUs)

PU6.3.1. Installing and Configuring MongoDB

Objective: To install and configure MongoDB on the local system and verify database service functionality.
Tasks:

1. Download and install MongoDB Community Edition or MongoDB Atlas.
2. Configure environment variables and start MongoDB service.
3. Use MongoDB Shell or Compass to create a test database.
4. Insert and view sample documents.

Expected Outcome:

- Students successfully install and connect to MongoDB locally or via cloud (Atlas).
- Students understand the basic MongoDB interface and data storage structure.

PU6.3.2. Connecting MongoDB with Express.js

Objective: To establish a connection between an Express.js server and MongoDB using Mongoose.
Tasks:

1. Create a Node.js project and install dependencies (express, mongoose, dotenv).
2. Connect the server to MongoDB using Mongoose.
3. Display a connection success/failure message in the console.

Expected Outcome:

- Students can integrate MongoDB with a Node.js application using Mongoose.
- Students understand connection strings and environment configuration.

PU6.3.3. Creating and Using Schemas and Models

Objective: To define data structure using Mongoose schemas and models for structured data handling.
Tasks:

1. Create a models folder and define a schema (e.g., studentSchema).
2. Add field validations (e.g., required, unique, min, max).
3. Generate a Mongoose model and test it by saving sample data.

Expected Outcome:

- Students can define schemas and models in Mongoose.
- Students understand schema validation and data consistency.

PU6.3.4. Performing CRUD Operations

Objective: To perform basic Create, Read, Update, and Delete operations using Mongoose models.
Tasks:

1. Implement POST, GET, PUT, and DELETE routes in Express.
2. Test all operations using Postman or Thunder Client.
3. Verify the results directly in MongoDB Compass.

Expected Outcome:

- Students can create RESTful routes connected to MongoDB.

- Students gain hands-on experience in managing data with Express and Mongoose.

PU6.3.5. Implementing Data Validation and Error Handling
Objective: To apply schema-based validation rules and handle errors gracefully in Express routes.
Tasks:
1. Add validation fields (e.g., required, minlength, maxlength) in schemas.
2. Implement try-catch blocks in CRUD routes for error handling.
3. Test with both valid and invalid data to observe responses.

Expected Outcome:
- Students can enforce validation rules in MongoDB collections.
- Students understand structured error handling in Express.js.

PU6.3.6. Deploying Node.js + MongoDB Application
Objective: To deploy the complete backend application to an online hosting platform.
Tasks:
1. Create a cloud MongoDB cluster using MongoDB Atlas.
2. Replace local connection string with the cloud connection string.
3. Deploy the Node.js app on Render or Vercel.
4. Verify API endpoints using Postman.

Expected Outcome:
- Students can deploy full backend applications connected to a live MongoDB database.
- Students understand environment configuration for production deployment.

Module 7: Deployment

7.1. Introduction
Deployment is the process of taking a web application from the development environment to a live production environment where users can access it online.
For MERN stack applications, deployment involves hosting the frontend (React) and backend (Node + Express) on servers, connecting them to a cloud database (MongoDB Atlas), and configuring all settings for performance and security.
This module helps learners understand the deployment process, environment variables, hosting options, and tools needed to make a web application live and reliable.

7.2. Learning Units (LUs)

LU7.2.1. Deployment to Heroku or Netlify
Deployment is the final stage of web development where an application is made available online for users. In this unit, learners will understand how to deploy both frontend (React.js) and backend (Node.js + Express) parts of a MERN application using popular cloud platforms such as Netlify for frontend hosting and Render or Heroku for backend hosting. Students will also explore GitHub Pages as an alternative for deploying static web projects.

7.2.1.1. Understanding Web Deployment
Deployment involves moving your web application from a local development environment to a remote server (cloud hosting) so that users can access it via a URL.
It ensures your app is accessible, scalable, and secure for real-world use.

Key Deployment Goals:
- Make the application available online.
- Ensure stable connection between frontend, backend, and database.
- Manage version updates using Git.
- Automate builds and error logging.

7.2.1.2. Tools & Libraries Used

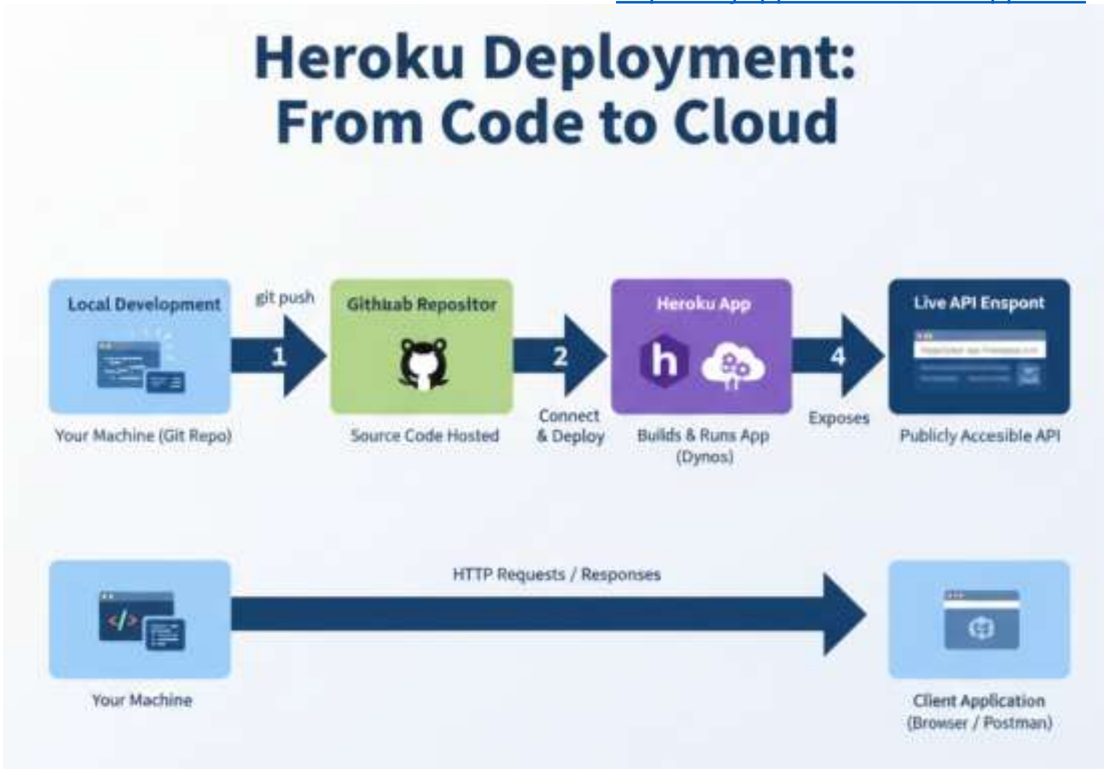
Tool / Platform	Purpose	Description
Netlify	Frontend Hosting	Ideal for React.js applications. Offers free hosting, CI/CD integration, and automatic deployment from GitHub.
Render / Heroku	Backend Hosting	Used for deploying Node.js + Express servers. Supports environment variables, scaling, and Git-based deployment.
GitHub Pages	Static Site Hosting	Suitable for HTML/CSS/JS or static React builds. Provides free hosting for personal or portfolio projects.
MongoDB Atlas	Cloud Database	Stores application data and connects seamlessly with deployed backend.

7.2.1.3. Steps for Deploying Backend (Heroku or Render)
A. Prepare Your App
- **i.** Ensure server.js or app.js has:
```
const PORT = process.env.PORT || 5000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```
- **ii.** Add "start": "node server.js" to package.json scripts.
- **iii.** Push your code to GitHub.

B. Deploy to Heroku
- **i.** Create a Heroku account → https://www.heroku.com
- **ii.** Install Heroku CLI and login (heroku login).
- **iii.** Initialize Git if not already done:
```
git init
git add .
git commit -m "initial commit"
```

iv. Create Heroku app:
    heroku create myapp-name
v. Deploy:
    git push heroku main
vi. Add environment variables using:
    heroku config:set MONGO_URI=your_connection_string

Result: Your backend API is live and accessible at https://myapp-name.herokuapp.com.



7.2.1.4. Steps for Deploying Frontend (Netlify)

i. Build the React app:
    npm run build
ii. Visit https://www.netlify.com and log in.
iii. Click "Add New Site → Import from Git" and connect your GitHub repo.
iv. Set:
    - Build Command: npm run build
    - Publish Directory**:** build/
v. Deploy and test your live frontend at the provided Netlify URL.

Result: Your React app is now hosted online, connected to your backend API.

7.2.1.5. Using GitHub Pages for Static Sites

For simple frontend-only websites:

1. Install the package:
   npm install gh-pages --save-dev
2. Add the following to package.json:
```
"homepage": "https://username.github.io/myapp",
"scripts": {
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}
```
3. Run:
   npm run deploy

The static site will be live at https://username.github.io/myapp.

7.2.1.6. Common Deployment Best Practices

- Always test locally before deployment.

- Keep API keys and credentials in environment variables.
- Use .gitignore to hide sensitive data.
- Regularly update and monitor your app post-deployment.
- Use tools like Postman to test API endpoints after going live.

7.3 Practical Units (PUs)

PU7.3.1. Deploying Backend API on Render or Heroku
Objective: Students will deploy their Node.js + Express backend API to Render or Heroku, making it publicly accessible through a live URL.
Tools & Technologies:
- Node.js, Express.js
- GitHub
- Render or Heroku
- MongoDB Atlas

Activity Steps:
1. Verify that your backend works locally (npm start).
2. Create a .env file and store environment variables (like MongoDB URI).
3. Initialize Git and push your backend code to GitHub.
4. Log in to Render.com or Heroku.com.
5. Create a New Web Service and connect your GitHub repository.
6. Set:
   - Build Command: npm install
   - Start Command: npm start
   - Add your MONGO_URI in Environment Variables.
7. Deploy and test your live backend endpoint (e.g., https://myapp.onrender.com/api/users).

Expected Outcome:
A live backend API that connects successfully to MongoDB Atlas and responds to HTTP requests.

PU7.3.2. Deploying Frontend (React App) on Netlify
Objective: Students will deploy the frontend of their MERN application using Netlify, connected to the previously deployed backend API.

Tools & Technologies:
- React.js
- Netlify
- GitHub

Activity Steps:
1. Ensure the API base URL in your frontend code points to the deployed backend.
2. Build your app using:
3. npm run build
4. Push your frontend code to GitHub.
5. Log in to Netlify → "Add New Site" → "Import from Git".
6. Set:
   - Build Command: npm run build
   - Publish Directory**:** build/
7. Click Deploy Site.
8. Test your live frontend using the Netlify-provided link.

Expected Outcome:
A live, fully functional React app that communicates with the deployed backend API.

PU7.3.3. Hosting Static Website using GitHub Pages
Objective:
To deploy a simple static website (HTML, CSS, JS) or React build folder using GitHub Pages.
Tools & Technologies:

- GitHub
- GitHub Pages

Activity Steps:

1. Create a GitHub repository and push your static site files (index.html, style.css, etc.).
2. Go to Repository Settings → Pages → Deploy from Branch.
3. Choose main branch and /root folder.
4. Save and wait for the live link.
5. For React projects, install:
6. npm install gh-pages --save-dev

Add in package.json:
```
"homepage": "https://username.github.io/myapp",
"scripts": {
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}
```
Then run:
```
npm run deploy
```

Expected Outcome:
A public GitHub Pages URL hosting your website, accessible to anyone online.

PU7.3.4. Connecting Frontend and Backend (Live Integration Test)
Objective: To test the complete integration of the deployed frontend, backend, and database.
Tools & Technologies:

- Netlify (Frontend)
- Render/Heroku (Backend)
- MongoDB Atlas

Activity Steps:

1. Open the deployed frontend link (Netlify).
2. Perform any form submission or action that calls the API (e.g., login, add data).
3. Monitor responses from the live backend API using browser DevTools → Network tab.
4. Check MongoDB Atlas to confirm data insertion or updates.
5. Fix any CORS or environment variable issues if they occur.

Expected Outcome:
A fully functional MERN stack application deployed online, communicating smoothly between all components.

Module 8: Entrepreneurship

8.1. Introduction

Entrepreneurship is the process of identifying opportunities, developing innovative ideas, and turning them into successful businesses or solutions. In today's digital world, tech entrepreneurship plays a vital role in driving innovation, solving problems, and creating economic growth.

For developers and IT professionals, understanding entrepreneurship helps transform technical skills into real-world applications — such as creating web platforms, startups, or SaaS (Software as a Service) products. This module will guide students through the fundamentals of entrepreneurship, idea generation, business models, and startup strategies.

Startup Ecosytem
Innovation
Teamwork
Big Idea
Business Model
Technology

8.2. Learning Units (LUs)

LU8.2.1. Introduction to Entrepreneurship

Entrepreneurship is the process of identifying opportunities, developing innovative solutions, and organizing resources to create and manage a business venture. It plays a vital role in driving economic growth, fostering innovation, and generating employment. In today's technology-driven world, entrepreneurship extends beyond traditional business—it includes tech startups, social enterprises, and digital innovations that solve real-world problems through creativity and strategy.

Entrepreneurs are visionaries who transform ideas into actionable business models. They take calculated risks, manage uncertainty, and bring together people, capital, and innovation to achieve success. Understanding entrepreneurship helps learners develop an entrepreneurial mindset, preparing them to become creators rather than just consumers of technology and opportunities.

8.2.1.1. The Concept of Entrepreneurship

Entrepreneurship combines innovation, initiative, and risk-taking to deliver new products or services that meet market needs. It involves recognizing a gap in the market, designing a solution, and turning it into a profitable venture.

Entrepreneurs are characterized by their ability to identify problems, develop creative ideas, and take responsibility for the outcomes of their ventures.

They are often seen as change-makers who challenge the status quo, introduce new technologies, and create social and economic impact.

8.2.1.2. Importance of Entrepreneurship in Today's Economy

Entrepreneurship is crucial in the modern digital economy for several reasons:

- **Job Creation:** New businesses generate employment opportunities.
- **Innovation:** Entrepreneurs drive technological progress and process improvements.
- **Economic Growth:** Successful ventures contribute to GDP and community development.
- **Social Impact:** Many modern entrepreneurs focus on sustainability and social well-being.
- **Technological Advancement:** Tech startups accelerate digital transformation across industries.

By promoting innovation and problem-solving, entrepreneurship becomes a key pillar of sustainable development and national competitiveness.

8.2.1.3. Types of Entrepreneurs

Entrepreneurs come from diverse backgrounds and can be categorized based on their goals and approaches:

Type	Description	Example

Small Business Entrepreneur	Focuses on local or small-scale operations.	Local bakery or digital agency.
Scalable Startup Entrepreneur	Builds ventures with high growth potential.	Tech startups (e.g., apps or SaaS).
Social Entrepreneur	Solves societal issues through innovation.	Non-profits using technology for education.
Innovative Entrepreneur	Introduces new technologies or processes.	Developers creating AI-based tools.
Imitative Entrepreneur	Adapts successful ideas to new markets.	Franchises or localized versions of apps.

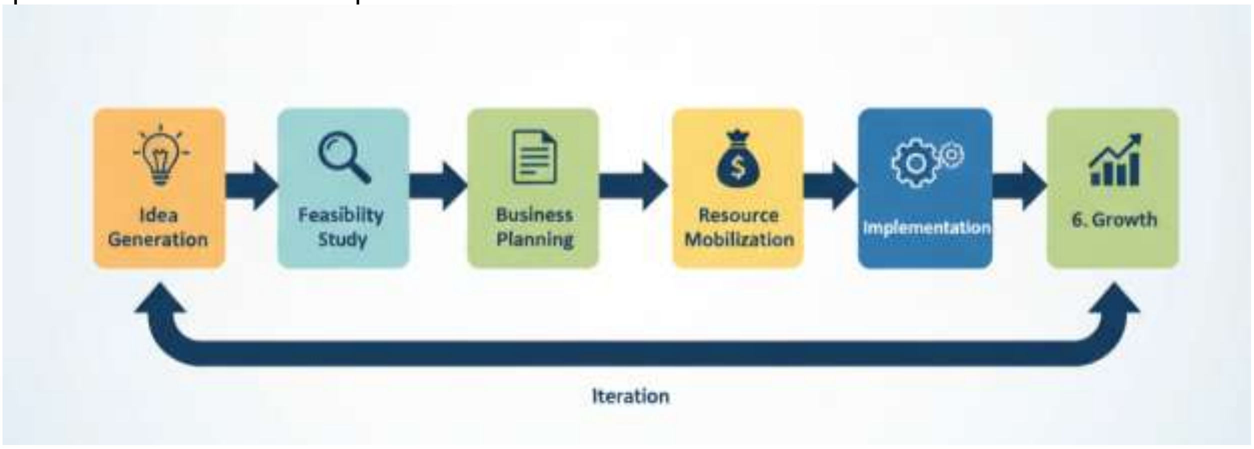
Each type requires a unique mindset, resource strategy, and risk appetite.

8.2.1.4. The Entrepreneurial Process

The entrepreneurial journey follows a structured process:

i. **Idea Generation:** Identifying opportunities and creative solutions.
ii. **Feasibility Analysis:** Evaluating market demand, resources, and potential challenges.
iii. **Business Planning:** Creating a roadmap for implementation.
iv. **Resource Mobilization:** Securing funding, partnerships, and human capital.
v. **Implementation:** Turning the plan into a functioning business.
vi. **Growth & Scaling:** Expanding operations and sustaining innovation.

This process encourages strategic thinking, adaptability, and long-term vision — essential qualities for modern developers and innovators.



8.2.1.5. The Role of Technology in Modern Entrepreneurship

In the digital age, technology empowers entrepreneurs to reach global audiences and innovate faster. Platforms like MERN Stack, cloud computing, and AI tools allow developers to build scalable, efficient, and secure products quickly.

Modern entrepreneurs rely on technology for:

- **Online presence and digital marketing**
- **E-commerce and online payment systems**
- **Data analytics for decision-making**
- **Remote collaboration and project management tools**

This synergy between technology and entrepreneurship creates new possibilities in every field — from education and healthcare to e-commerce and sustainability.

8.2.1.6. Entrepreneurial Mindset and Qualities

Successful entrepreneurs share a mindset built on curiosity, resilience, and adaptability. Key qualities include:

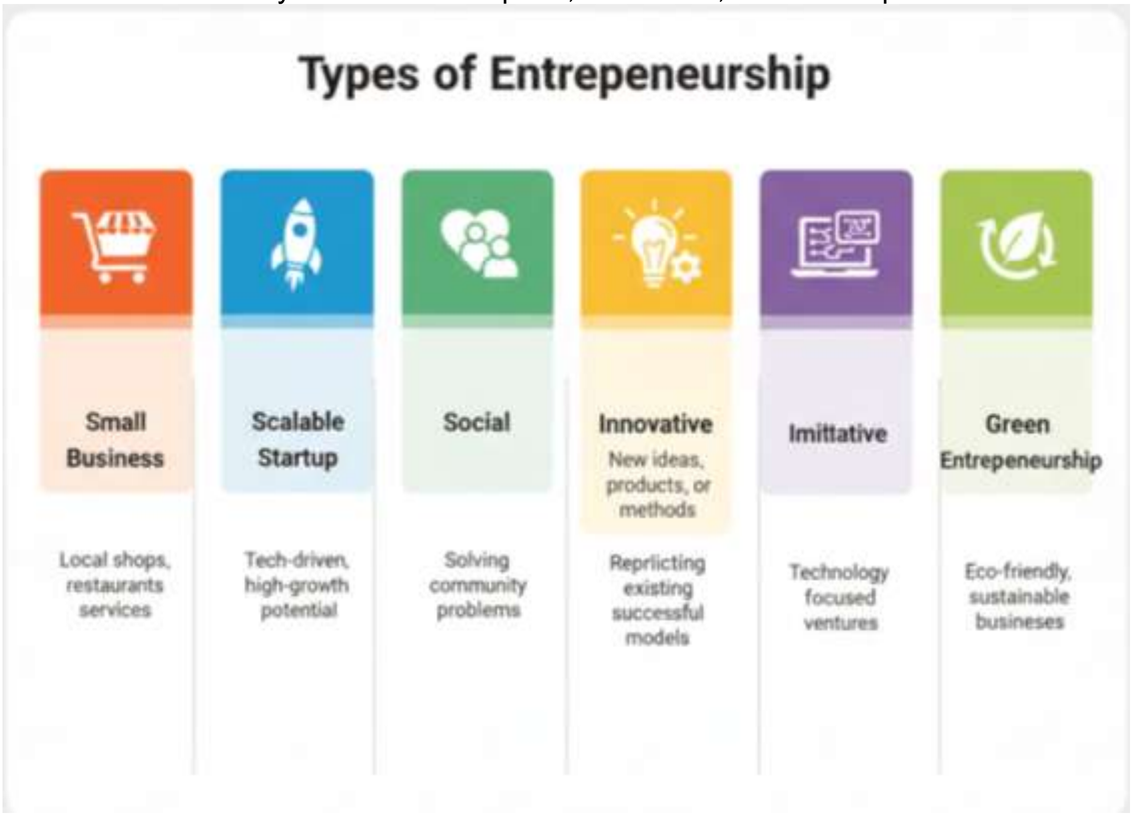
- **Creativity:** Thinking beyond conventional boundaries.
- **Risk Management:** Making informed decisions under uncertainty.
- **Leadership:** Guiding teams with confidence and empathy.
- **Persistence:** Overcoming challenges with determination.

- **Vision:** Seeing potential where others see problems.

Developing these traits prepares learners for both business and career success.

LU8.2.2. Types of Entrepreneurships

Entrepreneurship takes many forms, depending on the entrepreneur’s goals, vision, resources, and the type of value they aim to create. Understanding the different types of entrepreneurship helps learners recognize diverse pathways to starting and managing ventures — whether they are focused on profit, innovation, or social impact.


Types of Entrepeneurship
Small Business
Local shops, restaurants services
Scalable Startup
Tech-driven, high-growth potential
Social
Solving community problems
Innovative
New ideas, products, or methods
Reprlicting existing successful models
Imittative
Technology focused ventures
Green Entrepeneurship
Eco-friendly, sustainable busineses

Each type operates under unique motivations, business models, and risk levels, but all contribute to economic development and innovation in their own way.

8.2.2.1. Small Business Entrepreneurship

Small business entrepreneurship involves setting up and managing a business on a local or small scale. These businesses often serve community needs and provide employment opportunities.

Examples include local shops, small agencies, or online stores.

Key features:

- Limited capital investment
- Locally focused customer base
- Owner-managed operations
- Gradual growth through customer loyalty

8.2.2.2. Scalable Startup Entrepreneurship

Scalable startups are built with the goal of rapid expansion and high profitability. Entrepreneurs in this category often seek funding from investors to scale quickly and reach global markets.

Key features:

- Innovative ideas with high growth potential
- Technology-driven solutions (e.g., apps, SaaS)
- Focus on scalability and market disruption
- Venture capital or angel investment support

Example: Tech companies like Uber, Airbnb, or food delivery apps.

8.2.2.3. Social Entrepreneurship
Social entrepreneurs focus on solving societal problems through innovative and sustainable approaches. Their primary goal is social impact, not just profit.
Key features:
- Mission-driven ventures
- Solutions for education, environment, or healthcare
- Sustainable and community-oriented business models
- Focus on measurable social change

Example: NGOs using technology to promote education or health awareness.
8.2.2.4. Innovative Entrepreneurship
Innovative entrepreneurs develop new products, services, or technologies that revolutionize industries. They are driven by creativity and a desire to change existing processes or introduce groundbreaking solutions.
Key features:
- High investment in research and development
- Strong focus on creativity and originality
- Potential for global market disruption
- Risk-taking with long-term rewards

Example: Developers creating AI-based tools or renewable energy solutions.
8.2.2.5. Imitative Entrepreneurship
Imitative entrepreneurs adapt or modify successful ideas from existing markets and implement them in new contexts. Instead of inventing from scratch, they replicate proven models and localize them to fit customer needs.
Key features:
- Lower risk compared to innovative ventures
- Quick market entry with proven models
- Focus on customization for local markets
- Efficient use of existing ideas and systems

Example: Launching a regional version of a successful international e-commerce platform.
8.2.2.6. Technopreneurship
Technopreneurs use technology and innovation as the backbone of their businesses. This type of entrepreneurship merges IT, creativity, and business strategy to develop digital products and online services.
Key features:
- Technology-focused (web, mobile, AI, IoT)
- Global reach through digital platforms
- Low physical infrastructure needs
- Fast scalability and adaptability

Example: Developers launching SaaS applications, AI tools, or mobile platforms.
8.2.2.7. Green or Ecopreneurship
Green entrepreneurs build businesses around environmental sustainability. Their goal is to protect the planet while achieving profitability through eco-friendly solutions.
Key features:
- Focus on renewable energy, recycling, and sustainable practices
- Eco-conscious branding and business ethics
- Balancing profit with environmental responsibility
- Appeals to socially aware consumers

Example: Companies developing biodegradable packaging or clean energy products.

LU8.2.3. Business Idea Generation
Business idea generation is the creative process of identifying new business opportunities based on market needs, trends, or problems that need solutions. It is the foundation of entrepreneurship — a good idea is the first step toward building a successful business.

Entrepreneurs generate ideas through observation, brainstorming, innovation, and understanding customer pain points. The goal is to create ideas that are feasible, innovative, and marketable.

8.2.3.1. Sources of Business Ideas

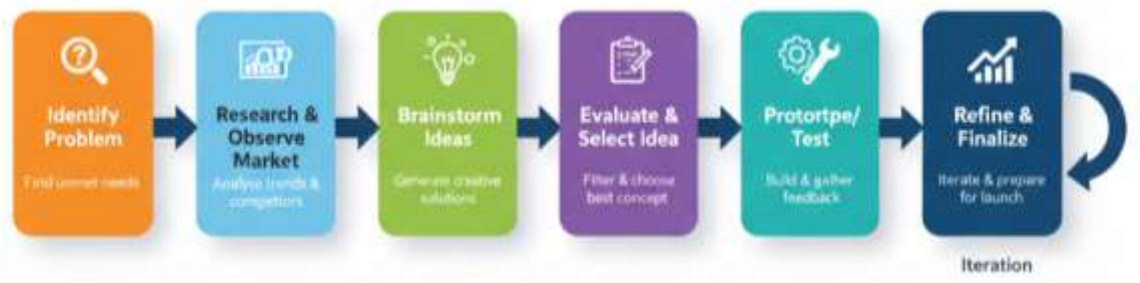
- i. **Market Gaps** – Identifying unmet needs in the market.
- ii. **Customer Feedback** – Listening to what customers want or complain about.
- iii. **Technological Trends** – Using new technologies to create better solutions.
- iv. **Personal Experiences** – Solving problems you face in everyday life.
- v. **Environmental Changes** – Adapting to social, political, or economic shifts.
- vi. **Research and Development (R&D)** – Innovation through experiments and creativity.

8.2.3.2. Idea Generation Techniques

- i. **Brainstorming:** Gathering a team to produce as many ideas as possible without judgment.
- ii. **Mind Mapping:** Visualizing ideas and their relationships in a diagram.
- iii. **SCAMPER Technique:** Modify existing ideas by Substituting, Combining, Adapting, Modifying, Putting to another use, Eliminating, or Reversing.
- iv. **SWOT Analysis:** Evaluating Strengths, Weaknesses, Opportunities, and Threats to identify viable ideas.
- v. **Observation Method:** Watching people's behaviors, habits, and problems in real-world settings.

8.2.3.3. Steps in Business Idea Generation

1. Identify a problem or need.
2. Gather information from market research.
3. Brainstorm potential solutions.
4. Evaluate and filter ideas based on feasibility.
5. Select the most promising idea.
6. Test and refine the idea before execution.


Identify Problem
Find unmet needs
Research & Observe Market
Analyze trends & competitors
Brainstorm Ideas
Generate creative solutions
Evaluate & Select Idea
Filter & choose best concept
Protortpe/ Test
Build & gather feedback
Refine & Finalize
Iterate & prepare for launch
Iteration

LU8.2.4. Business Planning and Strategy

Business planning and strategy are the blueprints for turning a web development idea into a sustainable business.

A clear plan defines what the business will do, who the target customers are, how it will make money, and how it will grow over time.

In the web development industry, strategic planning helps teams move beyond coding — toward building digital products or services that solve real-world problems and attract paying clients.

8.2.4.1. Importance of Business Planning in Web Development

A business plan:

- Provides clarity of goals (e.g., launching a web agency, SaaS platform, or e-commerce solution).
- Helps in financial planning and securing investments or clients.
- Defines target markets such as startups, schools, or small businesses.

- Guides team management and project workflows.
- Identifies competitive advantages like faster delivery, design quality, or niche specialization.

8.2.4.2. Key Components of a Web Development Business Plan

Component	Description	Example in Web Context
Executive Summary	Overview of your business and goals.	“We develop modern, responsive websites for local businesses.”
Market Analysis	Research competitors and customer needs.	Analyzing demand for e-commerce sites in local markets.
Services Offered	Define your core services.	Website design, app development, SEO, hosting, etc.
Target Audience	Identify who you’ll serve.	Small businesses, schools, or startups.
Revenue Model	How the business will earn money.	Subscription-based websites, project contracts, or retainers.
Marketing Strategy	How you’ll reach customers.	Using SEO, social media, and portfolio showcases.
Operational Plan	Workflow and team structure.	Frontend, backend, and QA teams using Agile or Scrum.
Financial Plan	Expected costs and profits.	Hosting fees, tools, salaries, client payments, etc.

8.2.4.3. Strategy Building in Web Development

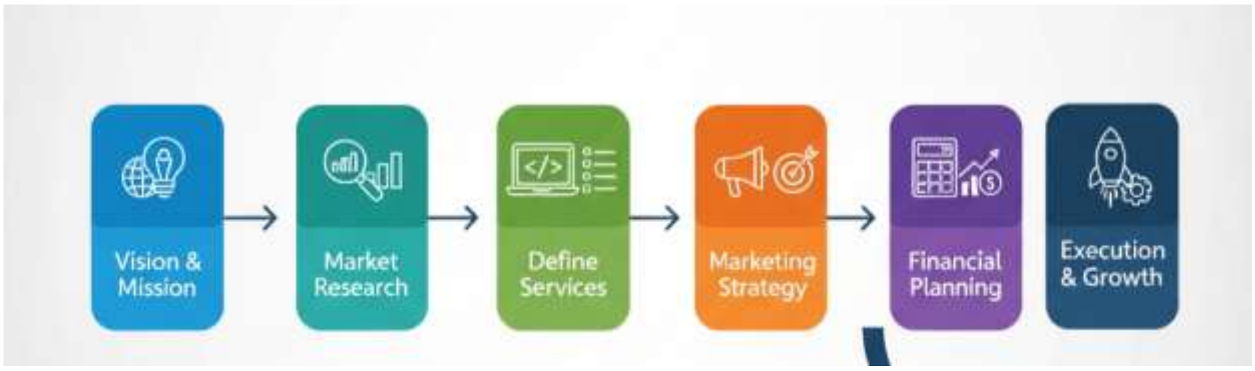
A **business strategy** focuses on how the web company will achieve a competitive edge.

Common Strategies:

i. Niche Specialization – Focus on a specific industry (e.g., education websites or health tech platforms).
ii. Quality and Performance – Offer speed, security, and modern UI/UX as selling points.
iii. Innovation – Use trending technologies (React, Next.js, Node.js, AI integrations).
iv. Client-Centric Approach – Offer after-delivery support and personalized service.
v. Scalability – Plan how projects and infrastructure can grow as clients increase.

8.2.4.4. Business Plan Execution Steps

i. Define Vision & Mission
ii. Set Short- and Long-Term Goals
iii. Conduct Market Research
iv. Develop a Service Portfolio
v. Create a Marketing Plan
vi. Set Up Tools & Technologies (VS Code, GitHub, Hosting Platforms)
vii. Monitor, Analyze, and Adjust Strategy



LU8.2.5. Financing Business

8.2.5.1. Introduction

Financing a business means arranging funds or capital to start, operate, and grow your enterprise.
In the context of web development or tech startups, financing helps entrepreneurs pay for:

- Development tools and software licenses
- Hosting and server costs
- Marketing and client acquisition
- Team salaries and operational expenses

A solid financial plan ensures that your web-based business runs smoothly — even before it starts generating consistent revenue.

8.2.5.2. Importance of Business Financing

Proper financing helps entrepreneurs:

- **Start strong** – with enough funds for initial setup.
- **Maintain cash flow** – to cover monthly expenses.
- **Invest in growth** – marketing, tools, and skill development.
- **Reduce risk** – by planning expenses and income carefully.
- **Build investor trust** – through transparent financial management.

Without adequate financing, even a great business idea may fail due to lack of sustainability.

8.2.5.3. Types of Business Financing

i. Self-Financing (Bootstrapping)

- Using personal savings or income to fund your project.
- Common for small web development agencies or freelancers.
- ✅ *Advantage:* Full control, no debt.
- ⚠️ *Limitation:* Limited growth due to personal fund limits.

ii. Friends and Family Support

- Borrowing or receiving investment from close contacts.
- ✅ *Advantage:* Easy to access, flexible repayment.
- ⚠️ *Limitation:* Can strain relationships if business struggles.

iii. Bank Loans and Microfinance

- Traditional method involving banks or lending institutions.
- Web developers may apply for SME loans or startup programs.
- ✅ *Advantage:* Larger capital for scaling.
- ⚠️ *Limitation:* Requires collateral and strong business plan.

iv. Angel Investors and Venture Capitalists

- Investors who fund startups with potential for high growth.
- Common for tech-based or SaaS startups.
- ✅ *Advantage:* Provides funds + mentorship.
- ⚠️ *Limitation:* May require sharing ownership (equity).

v. Crowdfunding

- Raising small amounts from a large number of people online (via Kickstarter, Indiegogo, etc.).
- ✅ *Advantage:* Good for innovative web app ideas.
- ⚠️ *Limitation:* Success depends on marketing reach.

vi. Government Grants and Startup Programs

- Many countries offer technology startup funds or innovation grants.
- ✅ *Advantage:* Non-repayable assistance.
- ⚠️ *Limitation:* Competitive and requires proposal submission.

Types of Business Financing
Self-Financing
Use personal savings.
Friends & Family
User inde dolor use use personal savings.
Bank Loans
Use lo and msith thackes and catyess
Angel Investors
Lse sdys anl dolor trime otups.
Crowfunding
Crowfunding contmenal of prcint sand sages.
Government Grants
corrrmenal dolos ourd sages.

8.2.5.4. Financial Planning in Web Development
Before seeking funds, developers should prepare:

i. **Budget Plan:** Estimate expenses (tools, domain, hosting, marketing).
ii. **Revenue Model:** Define how money will come in (clients, subscriptions, ads).
iii. **Profit Forecast:** Predict growth over 6–12 months.
iv. **Expense Tracking:** Monitor cash flow using tools like Excel or QuickBooks.

LU8.2.6. Entrepreneurship Challenges and Possible Solutions
Entrepreneurship is an exciting journey — full of creativity, independence, and opportunities for innovation.
However, entrepreneurs often face various challenges that test their determination, planning, and problem-solving skills
In the web development and tech startup world, these challenges can include financial constraints, competition, skill gaps, and managing clients effectively. Understanding these barriers and learning how to overcome them is key to building a sustainable business.

8.2.6.1. Common Challenges Faced by Entrepreneurs

i. Financial Constraints

- **Problem:** Lack of startup capital or inconsistent cash flow.
- **Example:** A new web agency struggles to afford premium hosting or marketing tools.
- **Solution:**
  - Start small using free/open-source tools.
  - Seek micro-loans, crowdfunding, or government startup grants.
  - Maintain a detailed financial plan and reduce unnecessary expenses.

ii. Competition and Market Saturation

- **Problem:** The web development market is highly competitive.
- **Example:** Many freelancers offer similar website development services at lower prices.
- **Solution:**
  - Focus on a niche (e.g., e-commerce, portfolio sites, or school systems).
  - Build a strong personal brand and showcase projects online (GitHub, Behance).
  - Offer excellent customer service to stand out.

iii. Managing Time and Multiple Tasks

- **Problem:** Entrepreneurs often juggle coding, marketing, and client communication simultaneously.
- **Solution:**
  - Use project management tools like Trello, Notion, or Asana.
  - Prioritize tasks using a weekly schedule.
  - Delegate non-technical work or use automation tools.

iv. Lack of Technical or Business Skills

- **Problem:** A developer may know coding but not marketing or finance.
- **Solution:**
  - Attend online workshops, bootcamps, or business courses.
  - Collaborate with others who have complementary skills.
  - Continuously learn new technologies and management practices.

v. Building and Retaining Clients

- **Problem:** Difficulty in finding consistent projects or maintaining long-term clients.
- **Solution:**
  - Create a professional portfolio and social media presence.
  - Focus on delivering quality and reliability to gain referrals.
  - Offer maintenance contracts for recurring revenue.

vi. Managing Stress and Motivation

- **Problem:** Pressure from deadlines, uncertainty, and failure can cause burnout.
- **Solution:**
  - Set realistic goals and celebrate small wins.
  - Maintain work-life balance.
  - Join entrepreneurship communities for support and motivation.

8.3 Practical Units (PUs)

PU8.3.1. Entrepreneurial Self-Assessment

Objective: To help learners identify their entrepreneurial strengths, weaknesses, and risk-taking abilities.

Activities:

- Conduct an Entrepreneurial Personality Assessment (using tools or self-reflection forms).
- Discuss traits like creativity, persistence, problem-solving, and leadership.
- Reflect on personal goals and motivations for becoming an entrepreneur.

Expected Outcome: Students understand their entrepreneurial mindset and readiness for starting a business.

PU8.3.2. Identify and Classify Types of Entrepreneurship

Objective: To recognize and differentiate between different forms of entrepreneurship (e.g., small business, scalable startup, social, and digital entrepreneurship).

Activities:

- Analyze 4–5 real-world startup case studies (including tech startups).
- Create a short presentation or chart classifying them into entrepreneurship types.
- Discuss why a web development firm fits under "digital or small business entrepreneurship."

Expected Outcome: Learners can classify ventures based on scale, purpose, and innovation.

PU8.3.3. Business Idea Generation Workshop
Objective: To develop innovative and feasible web or mobile app business ideas.
Activities:

- Brainstorm ideas using mind mapping or SCAMPER technique.
- Select one idea and briefly describe its purpose, target audience, and problem solved.
- Use digital tools like Miro, Canva, or Google Jamboard to design idea boards.

Expected Outcome: Students produce a list of potential startup ideas with one shortlisted concept ready for development.

PU8.3.4. Developing a Basic Business Plan (Web Development Context)
Objective: To create a simplified business plan for a web or software-based startup.
Activities:

- Use the Lean Canvas or Business Model Canvas framework.
- Define the business vision, value proposition, customer segment, pricing model, and marketing plan.
- Include cost estimation for hosting, domain, and marketing.

Expected Outcome: Learners prepare a basic, structured business plan that connects technical and business goals.

PU8.3.5. Financial Planning and Budget Estimation
Objective: To understand budgeting and financing methods for a startup project.
Activities:

- Create a basic financial sheet for a small web development business.
- Identify startup costs (equipment, software, marketing).
- Research and list possible funding sources: microloans, crowdfunding, grants, or investors.

Expected Outcome: Students understand cost planning and different funding options for launching a business.

PU8.3.6. Case Study — Entrepreneurship Challenges and Solutions
Objective: To analyze common startup challenges and apply problem-solving strategies.
Activities:

- Study a real or simulated case (e.g., “A small web agency struggling with client retention”).
- Identify the top 3 challenges and propose solutions.
- Present findings in group discussion or poster format.

Expected Outcome: Learners develop analytical and strategic thinking to handle real-world entrepreneurial problems.

Module 9: Environment

9.1. Introduction

The environment plays a crucial role in sustaining all forms of life on Earth. In the modern world, technology and the environment are deeply interconnected digital industries rely on energy, hardware, and cloud infrastructure that impact our planet's resources. For web developers and IT professionals, understanding environmental issues and adopting sustainable practices is essential to build a greener, more responsible digital future.

This module explores environmental challenges, human impacts, and sustainable technology practices helping learners develop awareness and responsibility toward environmental protection.

Module Objectives

By the end of this module, learners will be able to:

- Identify various types of environmental hazards and their causes.
- Understand how human activities, including digital and industrial work, affect the environment.
- Apply eco-friendly and sustainable practices in technology and workplaces.
- Recognize the importance of climate change awareness and conservation.
- Contribute actively toward environmental protection efforts in both personal and professional life.

9.2. Learning Units (LUs)

LU9.2.1. Introduction to Environmental Issues

The environment includes all living and non-living things that surround us — air, water, soil, plants, animals, and human beings. It provides essential resources like oxygen, food, and shelter. However, due to rapid industrialization, urbanization, and technological advancement, our environment faces serious threats that affect the balance of nature and the quality of human life.

Environmental issues are now a global concern and require immediate action from individuals, communities, and industries including the technology sector.

9.2.1.1. Major Environmental Issues

Some of the major environmental issues affecting our planet include:

i. **Air Pollution:** Emission of harmful gases and particles from vehicles, factories, and burning fuels.
ii. **Water Pollution:** Contamination of rivers, lakes, and oceans by industrial waste, chemicals, and plastics.
iii. **Land Pollution:** Improper disposal of waste, deforestation, and overuse of fertilizers and pesticides.
iv. **Global Warming:** Increase in Earth's temperature due to excessive greenhouse gas emissions.
v. **Deforestation:** Cutting down forests for agriculture, urbanization, and industrial use.
vi. **Loss of Biodiversity:** Disappearance of various plant and animal species due to habitat destruction.
vii. **E-Waste Generation:** Disposal of outdated electronic devices contributing to toxic waste buildup.

9.2.1.2. Environmental Issues in the Digital/IT Context

The digital world also contributes to environmental problems in unique ways:

- **Energy Consumption:** Data centers and servers consume large amounts of electricity, increasing carbon emissions.
- **E-Waste:** Frequent disposal of mobile phones, computers, and other devices creates toxic waste.
- **Non-Sustainable Production:** Manufacturing tech products requires rare minerals and chemicals harmful to the environment.

- **Carbon Footprint:** Every online activity — streaming, cloud computing, or storing data — uses power that contributes to CO_2 emissions.

Thus, web developers, IT professionals, and students must be aware of how their work impacts the planet and practice green computing and sustainable digital habits.

9.2.1.3. How We Can Help

- Adopt energy-efficient devices and servers.
- Choose eco-friendly hosting providers using renewable energy.
- Design lightweight, optimized websites that consume less data and energy.
- Practice digital minimalism — avoid unnecessary digital waste and storage.
- Support awareness campaigns promoting environmental responsibility.

Environmental Issues in the Digital Age
Air Pollution
Manufacturing electronics
Water Pollution
Toxic chemicals in water
Global Warming
Increased carbon footprint
E-Waste
Obslorite devices in landfalls
Carbon Emissions
Data centers & cloud computing
Resource Depletion
Rare earth minerals

LU9.2.2. Type of Environmental Hazard

Environmental hazards are natural or human-made events that threaten the health of humans, animals, and the ecosystem. These hazards can cause physical damage, affect biodiversity, and disrupt natural processes. Understanding different types of hazards helps us take preventive measures and design sustainable solutions — even in technology-related fields like data centers, urban infrastructure, and manufacturing.

9.2.2.1. Types of Environmental Hazards

Environmental hazards can be classified into four main categories:

i. Natural Hazards

These occur due to natural processes of the Earth and are often beyond human control. **Examples:**

- Earthquakes
- Floods
- Volcanic eruptions
- Cyclones and hurricanes
- Tsunamis
- Landslides

Impact Loss of life, property damage, disruption of ecosystems, and destruction of infrastructure.

ii. Man-Made (Anthropogenic) Hazards

These are caused by human activities such as industrialization, deforestation, and pollution.
Examples:

- Industrial waste and chemical spills
- Air and water pollution
- Nuclear accidents (e.g., Chernobyl, Fukushima)
- Oil spills
- Urban waste and e-waste accumulation

Impact: Toxic contamination, soil degradation, health issues, and long-term damage to natural resources.

iii. Biological Hazards

These arise from organisms or biological processes that can harm human health and the environment.
Examples:

- Infectious diseases (e.g., COVID-19, malaria)
- Bacterial contamination in food and water
- Invasive species destroying local ecosystems

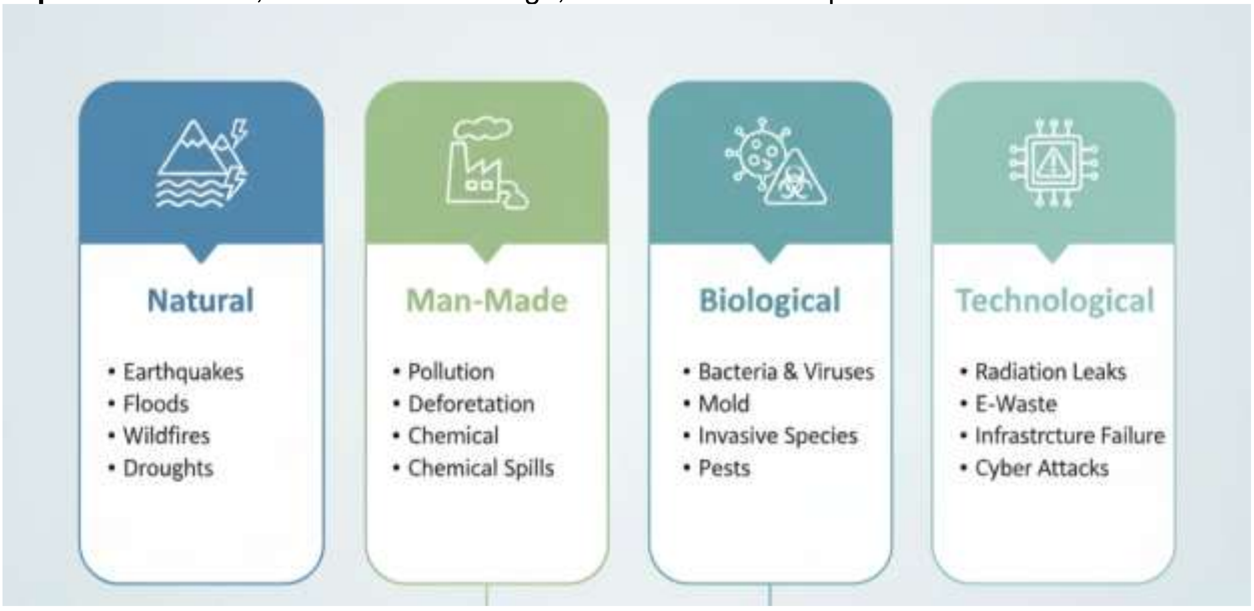
Impact: Health risks, reduced agricultural productivity, and ecosystem imbalance.

iv. Technological Hazards

These are linked to failures or misuse of technology and infrastructure.
Examples:

- Power plant explosions
- Data center fires
- Cyberattacks on critical infrastructure
- Industrial accidents

Impact: Loss of life, infrastructure damage, and economic disruption.



9.2.2.3. Environmental Hazards in IT and Web Development Context

Even the technology industry contributes to environmental hazards in indirect ways:

- E-waste from outdated hardware releases hazardous materials.
- Energy-intensive data centers contribute to carbon emissions.
- Non-biodegradable materials in tech manufacturing pollute soil and water.
- Server overheating or malfunction can cause localized hazards or fire risks.

Thus, developers and organizations should prioritize green technology and sustainable hardware disposal.

LU9.2.3. The Impact of Human Activity on the Environment

Human activities have significantly altered the Earth's natural systems. From industrialization to deforestation, and from urbanization to modern technology, nearly every human action leaves an environmental footprint. Understanding these impacts is crucial for developing sustainable practices and reducing further harm to the planet.

9.2.3.1. Deforestation

Forests are being cleared for agriculture, urban expansion, and industrial purposes. This reduces biodiversity and affects global carbon balance.

Impact:

- Loss of animal and plant habitats
- Increased carbon dioxide levels
- Soil erosion and loss of fertility
- Disruption of the water cycle

9.2.3.2. Industrialization

Rapid industrial growth increases production but also releases pollutants into the air, water, and soil.

Impact:

- Air pollution from factory emissions
- Water pollution from industrial waste
- Land degradation from toxic dumping
- Health issues in nearby populations

9.2.3.3. Urbanization

The expansion of cities leads to increased construction, energy use, and waste generation.

Impact:

- Increased air and noise pollution
- Strain on water and energy resources
- Reduction in green spaces
- Rise in temperature (urban heat islands

9.2.3.4. Pollution

Human activities generate pollutants that contaminate the environment.

Types and Effects:

- **Air Pollution:** Caused by vehicles, factories, and burning fossil fuels → leads to respiratory diseases and climate change.
- **Water Pollution:** Dumping of waste and chemicals into rivers → harms aquatic life.
- **Soil Pollution:** Use of pesticides and plastics → affects agriculture and food safety.
- **Noise Pollution:** From industries and transport → causes stress and hearing issues.

9.2.3.5. Climate Change

Excessive greenhouse gas emissions (like CO_2 and methane) trap heat in the atmosphere, causing global warming.

Impact:

- Melting of glaciers and polar ice
- Rising sea levels
- Extreme weather conditions (droughts, floods)
- Loss of biodiversity

9.2.3.6. E-Waste and Technological Impact

Improper disposal of electronic devices releases toxic materials into the environment.

Impact:

- Hazardous metals (lead, mercury) pollute soil and water
- Health problems for workers handling e-waste
- Energy consumption in data centers contributes to global carbon emissions

Sustainable Practices:

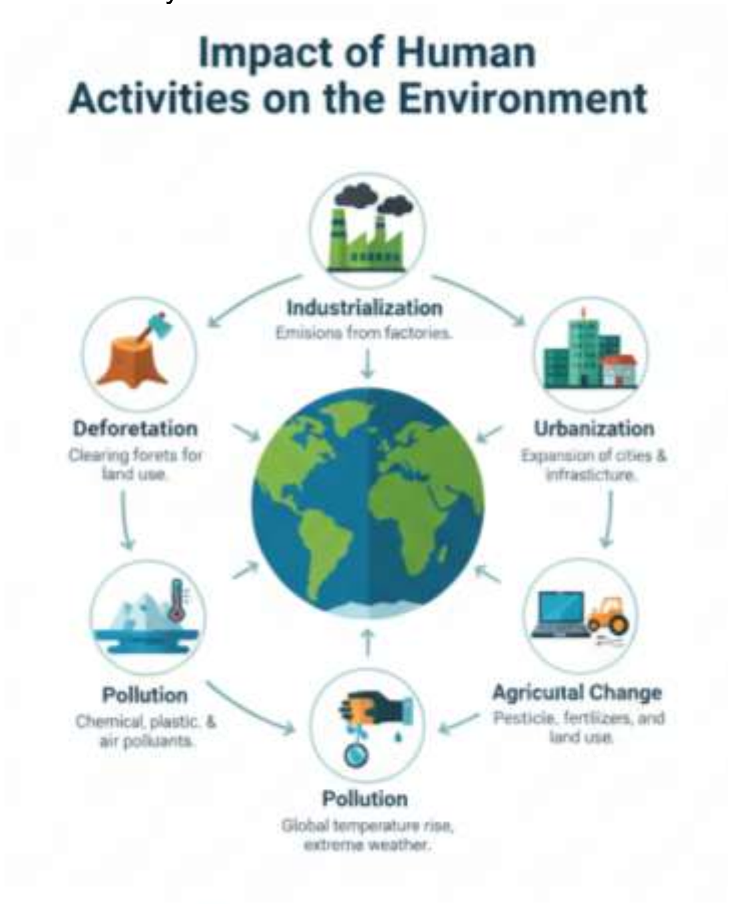
- Recycling old electronics
- Using energy-efficient hardware
- Adopting green hosting and sustainable software practices

9.2.3.7. Agricultural Practices

Modern farming relies heavily on chemical fertilizers and pesticides.

Impact:

- Water pollution through runoff
- Soil degradation and loss of nutrients
- Reduction in biodiversity due to monoculture


Impact of Human Activities on the Environment
Industrialization
Emisions from factories.
Deforetation
Clearing forets for land use.
Urbanization
Expansion of cities & infrastincture.
Pollution
Chemical, plastic, & air polluants.
Agricultal Change
Pesticie, fertlizers, and land use.
Pollution
Global temperature rise, extreme weather.

LU9.2.4. Conservation and Sustainability

Conservation and sustainability are key principles aimed at protecting natural resources and ensuring their availability for future generations. As human activities continue to affect ecosystems, it is essential to adopt responsible practices that minimize environmental impact while maintaining economic and social well-being. In web development and the tech industry, sustainability also includes energy-efficient practices and environmentally friendly digital solutions.

9.2.4.1. Conservation: Definition and Importance

Conservation refers to the responsible management and protection of natural resources—such as water, forests, soil, and wildlife—to prevent overuse, depletion, or destruction.

Objectives of Conservation:

- Preserve biodiversity and natural habitats
- Maintain ecological balance
- Ensure resources for future generations
- Reduce waste and pollution

Examples:

- Protecting endangered species through wildlife sanctuaries and national parks
- Implementing reforestation and afforestation programs
- Controlling overfishing and illegal logging
- Promoting water conservation and soil management

9.2.4.2. Sustainability: Definition and Principles

Sustainability means meeting present needs without compromising the ability of future generations to meet theirs. It integrates environmental, economic, and social dimensions—often represented as the "Three Pillars of Sustainability."

Three Pillars:

i. **Environmental Sustainability** – Protecting ecosystems, reducing pollution, and conserving natural resources.
ii. **Economic Sustainability** – Ensuring stable growth and efficient use of resources without damaging the environment.
iii. **Social Sustainability** – Promoting equity, health, education, and quality of life for all communities.

9.2.4.3. Sustainable Development Practices

To achieve sustainability, practical steps must be taken at personal, community, and industrial levels:

i. **Renewable Energy Use:** Switching to solar, wind, or hydro power reduces dependence on fossil fuels.
ii. **Waste Management:** Encouraging recycling, composting, and responsible disposal of electronic waste.
iii. **Green Technology:** Developing energy-efficient hardware, optimizing data centers, and designing eco-friendly digital products.
iv. **Sustainable Agriculture:** Using organic farming methods, crop rotation, and natural fertilizers to preserve soil health.
v. **Water and Energy Conservation:** Installing low-energy devices, fixing leaks, and practicing mindful consumption.

9.2.4.4. Role of Individuals and Developers

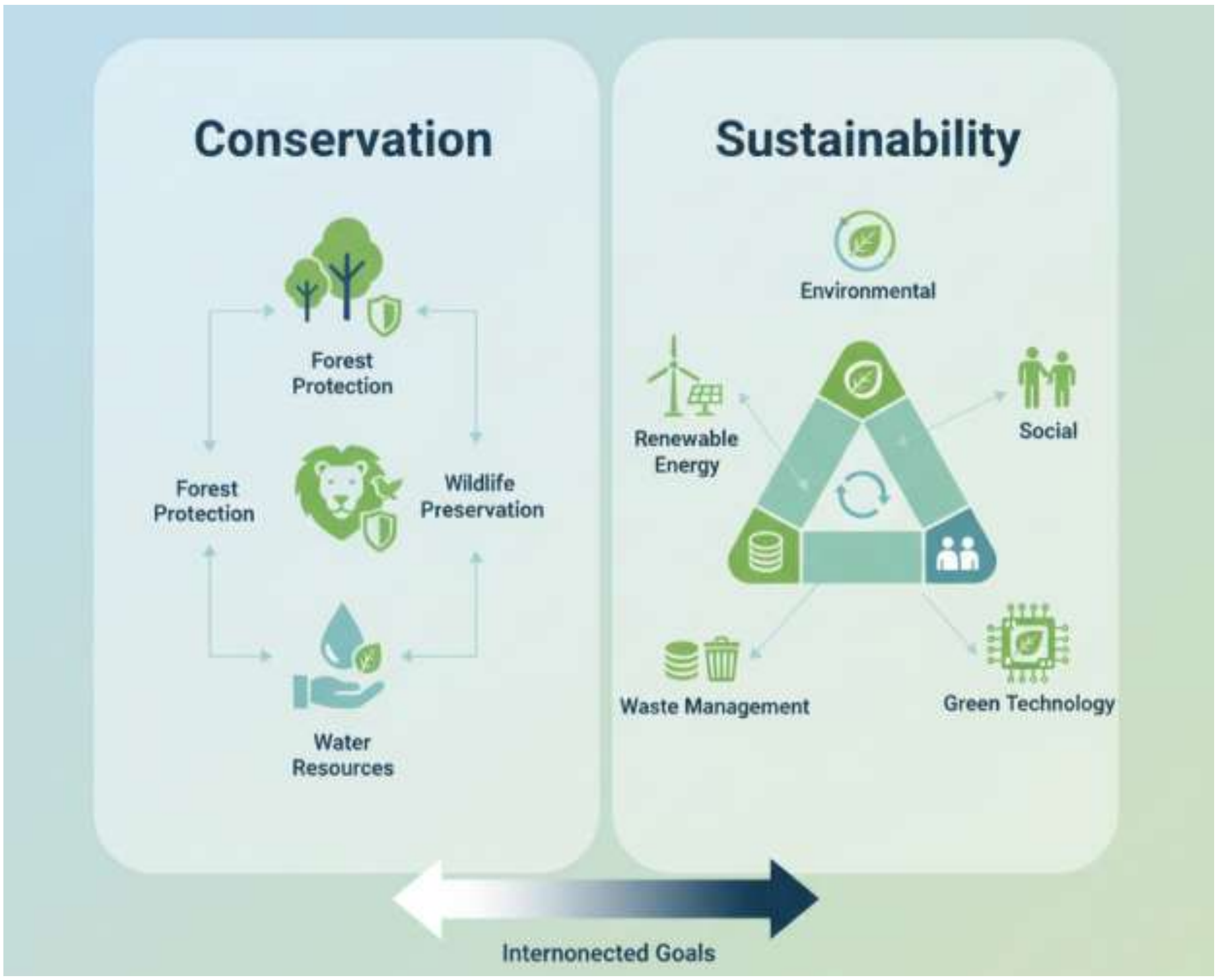
Even small actions can make a big difference. In the context of web and software development:

- Use green hosting services powered by renewable energy.
- Optimize websites for energy efficiency (lightweight designs and clean code).
- Reduce digital waste by removing unused data and optimizing storage.
- Educate teams and users about environmental awareness in digital practices.

9.2.4.5. Conservation and Sustainability in Action

Examples Worldwide:

- **The Paris Agreement (2015):** Global commitment to reduce carbon emissions.
- **UN Sustainable Development Goals (SDGs):** Promote responsible consumption, climate action, and life on land/water.
- **Recycling Programs in Tech:** Companies like Apple and Dell reuse old components and recycle e-waste.



LU9.2.5. Climate Change and Its Effects
Climate change refers to long-term alterations in global or regional climate patterns, primarily caused by human activities such as the burning of fossil fuels, deforestation, and industrial emissions. These actions increase greenhouse gases (GHGs) in the atmosphere, trapping heat and leading to a rise in the Earth's average temperature — a phenomenon known as global warming.
Understanding climate change and its effects is essential for developing strategies that promote environmental resilience, sustainable development, and global cooperation.

9.2.5.1. Causes of Climate Change
a. Greenhouse Gas Emissions
- The primary cause of climate change is the buildup of greenhouse gases such as carbon dioxide (CO_2), methane (CH_4), and nitrous oxide (N_2O).
- These gases trap heat in the atmosphere, creating the greenhouse effect, which warms the Earth's surface.

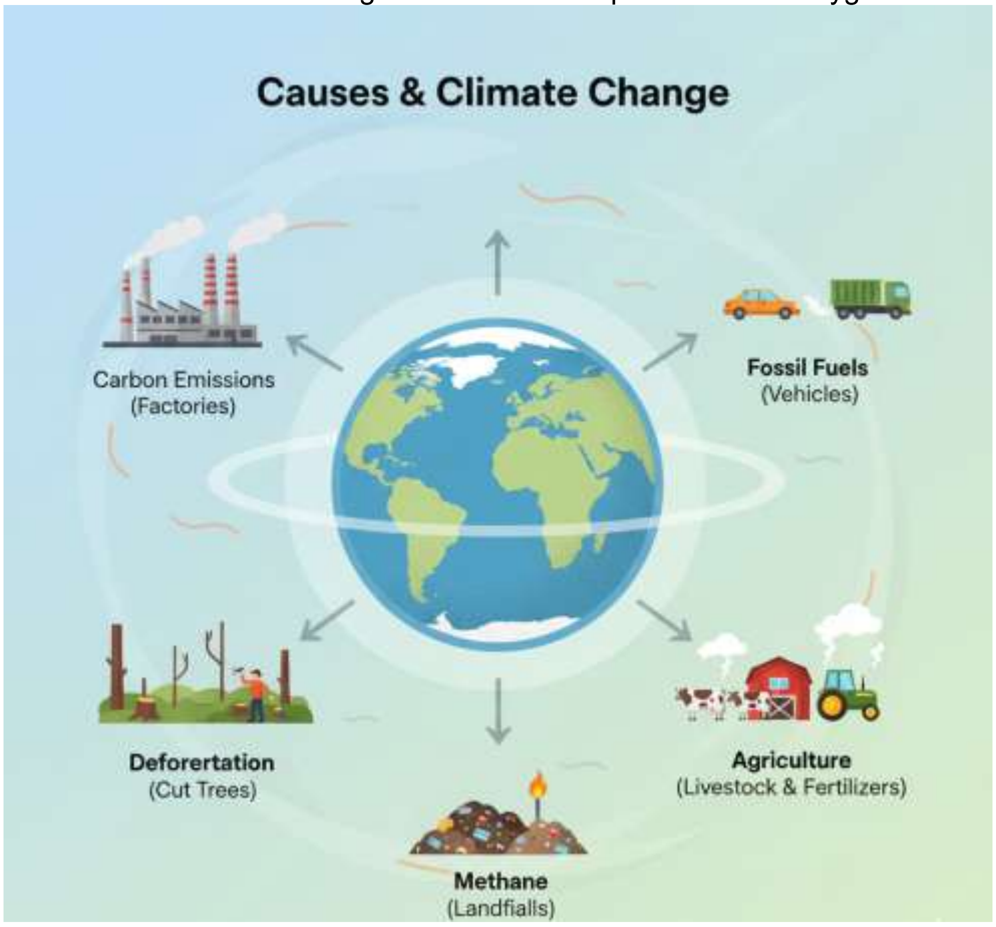
b. Deforestation
- Trees absorb CO_2; cutting them down releases stored carbon and reduces the planet's capacity to balance atmospheric gases.

c. Industrialization and Energy Use
- Burning coal, oil, and gas for electricity, manufacturing, and transportation releases large quantities of GHGs.

d. Agriculture
- Livestock farming produces methane; use of fertilizers releases nitrous oxide, both potent greenhouse gases.

e. Waste Management Issues

- Landfills emit methane as organic waste decomposes without oxygen.



9.2.5.2. Effects of Climate Change

a. Rising Global Temperatures

- Average global temperatures have increased significantly over the past century, resulting in melting glaciers, extreme heatwaves, and shifting weather patterns.

b. Melting Polar Ice and Rising Sea Levels

- Ice caps and glaciers are melting at an accelerated rate, causing sea levels to rise and threatening coastal communities.

c. Extreme Weather Events

- Hurricanes, floods, droughts, and wildfires are becoming more frequent and severe due to changing climate conditions.

d. Disruption of Ecosystems

- Many animal and plant species are losing their natural habitats, leading to extinction and biodiversity loss.

e. Impact on Agriculture and Food Security

- Irregular rainfall and changing temperatures affect crop yields, livestock health, and food supply chains.

f. Human Health Risks

- Increased temperatures and pollution contribute to heat stress, respiratory diseases, and the spread of vector-borne illnesses.

9.2.5.3. Global and Local Consequences

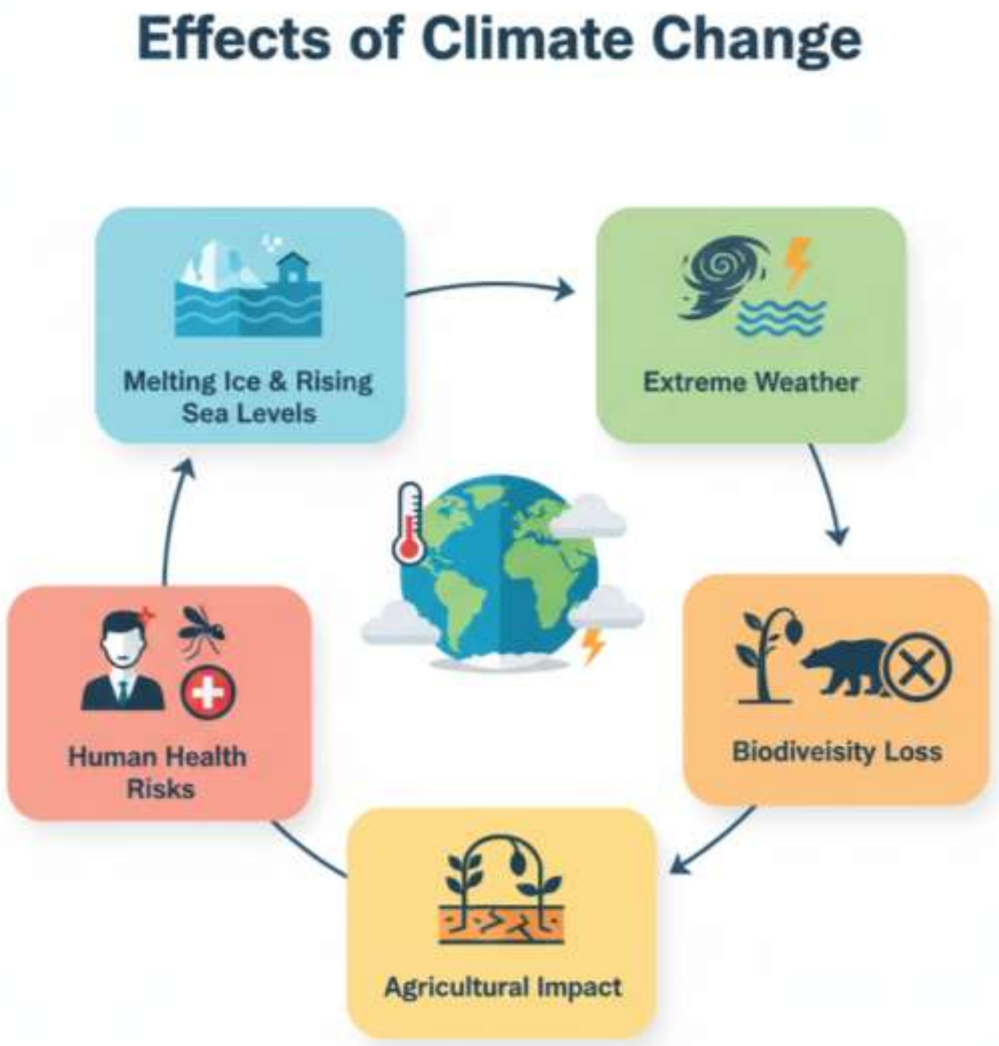
a. Global Consequences:

- Melting of Arctic ice, desertification, and coral bleaching.
- Migration crises due to uninhabitable regions.
- Economic losses from damage to infrastructure and agriculture.

b. Local Consequences (South Asia/Pakistan Example):

- Intense heatwaves and water scarcity.

- Glacial melting in northern regions causing floods.
- Disrupted monsoon patterns affecting agriculture.



9.2.5.4. Mitigation and Adaptation Strategies

Mitigation focuses on reducing or preventing the emission of greenhouse gases:

- Switching to renewable energy (solar, wind, hydro).
- Enhancing public transport and reducing fossil fuel dependency.
- Reforestation and afforestation initiatives.
- Energy-efficient buildings and sustainable urban planning.

Adaptation focuses on adjusting to the effects of climate change:

- Building flood defenses and drought-resistant crops.
- Establishing early warning systems for natural disasters.
- Promoting climate education and awareness.
- Supporting international agreements like the Paris Climate Accord.

9.2.5.5. Role of Technology and Digital Solutions

Digital technology plays an important role in monitoring and combating climate change:

- **Data Analytics & AI:** Predict weather patterns and track emissions.
- **IoT Sensors:** Monitor air and water quality.
- **Green Computing:** Reduce carbon footprint through energy-efficient systems.
- **Awareness Platforms:** Mobile and web apps that educate and promote sustainable behavior.

LU9.2.6. How to Contribute to Environmental Protection

Environmental protection is everyone's responsibility — individuals, organizations, and governments all play crucial roles in preserving natural resources and maintaining ecological balance. In the context of web development and technology, even digital activities have environmental impacts, such as energy consumption, electronic waste, and carbon emissions from data centers.

By adopting sustainable habits and eco-friendly digital practices, individuals and professionals can help reduce these negative effects and contribute to a greener, more sustainable planet.

9.2.6.1. Individual Contributions

Small actions at the personal level can collectively make a big difference.

a. Reduce, Reuse, Recycle (3Rs):

- Reduce waste by minimizing single-use plastics and unnecessary packaging.
- Reuse materials and products to extend their life cycle.
- Recycle paper, metal, glass, and electronic waste properly.

b. Save Energy:

- Turn off lights, fans, and devices when not in use.
- Use energy-efficient appliances and LED bulbs.
- Opt for laptops over desktops for lower energy consumption.

c. Use Sustainable Transportation:

- Prefer walking, cycling, or public transport over private cars.
- Use carpooling apps or electric vehicles to reduce carbon emissions.

d. Plant Trees and Support Green Spaces:

- Participate in tree-planting drives.
- Support local parks, gardens, and reforestation programs.

e. Practice Water Conservation:

- Fix leaks, use low-flow taps, and avoid water waste in daily activities.

9.2.6.2. Environmental Protection in the Workplace

Professionals, including those in the tech and web development sectors, can contribute through workplace initiatives.

a. Go Paperless:

- Use digital documents and cloud storage instead of printed materials.

b. Green Computing:

- Use energy-efficient servers and power management settings.
- Optimize software to reduce processing power and energy use.

c. E-Waste Management:

- Properly recycle old computers, phones, and cables.
- Avoid disposing of electronics in general waste bins.

d. Remote Work Practices:

- Encourage online meetings and hybrid work to reduce travel emissions.

e. Eco-Friendly Office Design:

- Implement natural lighting, ventilation, and indoor plants to reduce energy use and improve air quality.

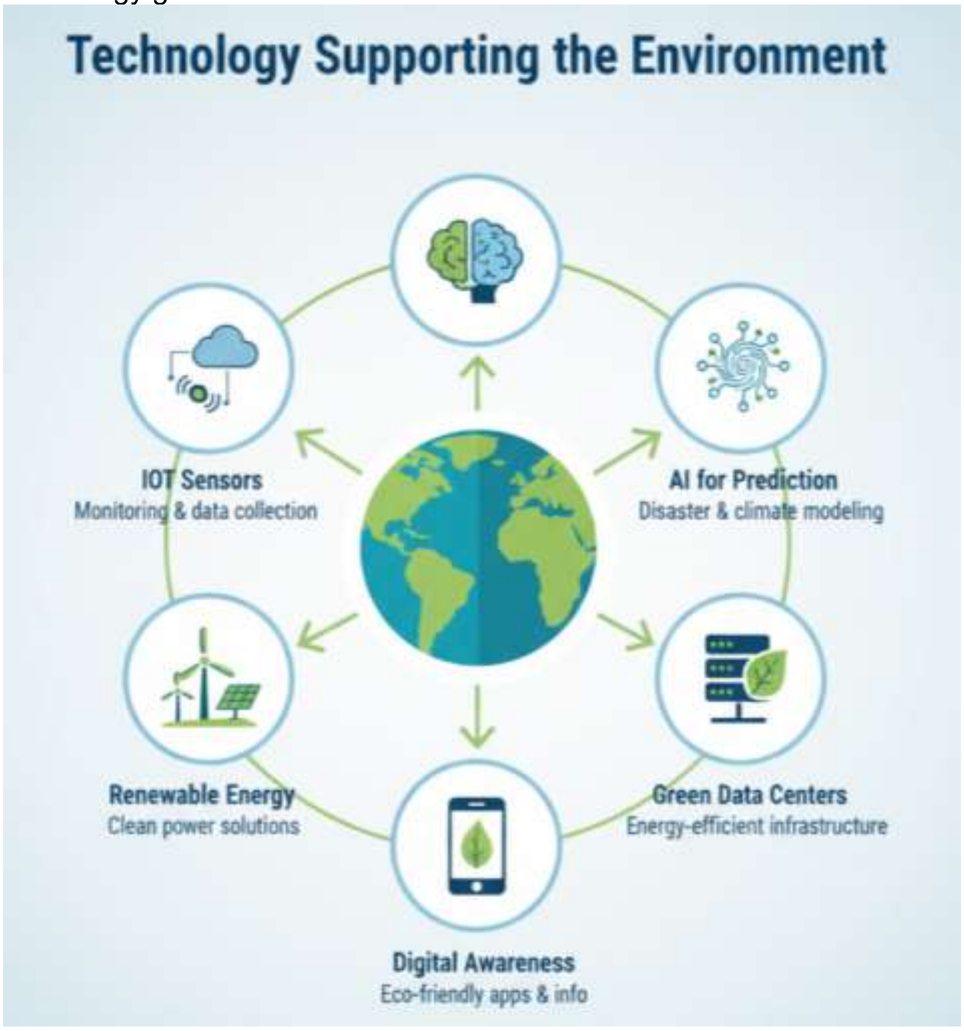
Everyday Ways
Protect Environment
Save Energy
Reduce electricity use
Recycle & Reuse
Sort waste, upycle
Recycle & Reuse
Sort waste refstuction
Go Paperless
Digital documents
Use Green Transport
Walk, bike, or carpool

9.2.6.3. Digital Sustainability (For Web Developers)
In the web and app development field, digital sustainability focuses on minimizing the carbon footprint of online platforms.
a. Optimize Websites for Efficiency:
- Reduce image sizes and server requests to consume less energy.
- Use green hosting services powered by renewable energy.

b. Promote Awareness through Design: Develop websites and apps that educate users about sustainability.
c. Use Dark Mode and Low-Data Designs: Implement features that reduce device power usage and data transfer.
d. Adopt Cloud Services Responsibly: Choose providers with carbon-neutral operations (like Google Cloud or AWS Sustainability).
9.2.6.4. Community and Global Actions
a. Volunteer for Environmental Campaigns: Join local or international initiatives such as beach cleanups, recycling drives, or awareness workshops.
b. Support Sustainable Brands: Choose companies that follow ethical sourcing, fair trade, and eco-friendly production.
c. Advocate for Policy Change: Encourage policymakers to adopt laws promoting renewable energy, conservation, and waste management.
d. Educate Others: Spread awareness through social media, seminars, or workshops about sustainable living and responsible digital practices.

9.2.6.5. The Role of Technology in Environmental Protection
Technology can both harm and help the environment — depending on how it's used. When applied responsibly, it becomes a key tool in conservation.

- **IoT Sensors:** Monitor pollution levels, water quality, and forest health.
- **Artificial Intelligence (AI):** Predict natural disasters and optimize energy use.
- **Data Analytics:** Track carbon emissions and suggest eco-friendly strategies.
- **Renewable Energy Technologies:** Solar panels, wind turbines, and smart grids for clean energy generation.



9.3. Practical Units (PUs)

PU9.3.1. Conduct an Environmental Awareness Survey
Objective: To understand the level of environmental awareness among students or employees within a department or organization.
Task:

- Design a Google Form or HTML-based survey form that includes questions on waste management, water usage, energy conservation, and sustainable practices.
- Collect and analyze responses to identify key awareness gaps.

Tools & Technologies: Google Forms / HTML + JavaScript / Excel for data analysis.
Expected Outcome: A summarized report highlighting awareness levels and suggested improvements.

PU9.3.2. Develop a "Green Tips" Web Page
Objective: To create a web page promoting eco-friendly habits and sustainable living tips.
Task:

- Design a responsive HTML/CSS web page titled “Go Green”.
- Include daily eco-tips (e.g., saving water, reducing electricity, recycling, etc.).
- Add visuals or icons for each tip using free resources like Flaticon or Canva.

Tools & Technologies: HTML, CSS, JavaScript, Canva/Flaticon.
Expected Outcome: A simple, visually appealing web page spreading environmental awareness.

PU9.3.3. Carbon Footprint Estimator App
Objective: To design a small web-based calculator that estimates a user’s carbon footprint based on energy, travel, and lifestyle habits.
Task:

- Build an interactive JavaScript form where users enter their daily activities (e.g., vehicle use, electricity consumption).
- Display the estimated carbon emissions and show tips to reduce them.

Tools & Technologies: HTML, CSS, JavaScript.
Expected Outcome: A working prototype that educates users about their environmental impact and encourages responsible behavior.

PU9.3.4. E-Waste Collection Awareness Poster
Objective: To promote awareness about the proper disposal of electronic waste (e-waste).
Task:

- Design a digital poster or infographic showing how to dispose of e-waste safely.
- Include statistics about global e-waste problems and nearby recycling centers.

Tools & Technologies: Canva / Adobe Express / PowerPoint / Figma.
Expected Outcome: A creative poster encouraging proper e-waste management in educational and workplace environments.

PU9.3.5. Research Report on “Green Computing Practices”
Objective: To explore eco-friendly computing strategies that reduce digital pollution.
Task:

- Research and write a short report (2–3 pages) on how companies apply green computing (e.g., Google, Microsoft, Apple).
- Include topics such as server optimization, renewable energy data centers, or sustainable hardware design.

Tools & Technologies: Word / Google Docs / PDF format for submission.
Expected Outcome: A professional report summarizing sustainable technology practices in the IT industry.

PU9.3.6. Environmental Data Visualization Dashboard
Objective: To visualize environmental data using digital tools for better understanding and awareness.
Task:

- Use Chart.js or Google Charts to create interactive visualizations such as:
    - Air quality index over time.
    - Energy consumption trends.
    - Waste reduction progress.
- Display graphs in a single dashboard-style web page.

Tools & Technologies: HTML, CSS, JavaScript (Chart.js / Google Charts).
Expected Outcome: A functional environmental data dashboard that supports data-driven awareness and education.

PU9.3.7. “Tech for Nature” Mini Campaign
Objective: To apply web and social media tools for environmental advocacy.
Task:

- Create a small online campaign using a simple website, blog post, or social media content promoting Earth protection themes.

- Use visuals, infographics, or short educational videos.

Tools & Technologies: WordPress / Blogger / HTML site / Canva / social media Tools.
Expected Outcome: A creative digital campaign showing how technology can be used to protect the environment.

Trainer Qualification Level

Qualification Level of trainer	Qualification / Certification	Purpose / Importance
Minimum	• 16 years of education in Computer Science, Software Engineering, IT, or related field • Basic certifications/online courses (Coursera, Udemy, edX)	Provides foundational knowledge and technical skills for effective training. Preferred
Mandatory	• 18 years of education / specialization in relevant field • Expertise in Full-Stack Web or Mobile App Development • Industry experience with hands-on projects	Ensures advanced expertise and practical experience for high-quality, industry-relevant instruction.

Job Opportunities

- Front-End Developer: Builds user-facing website interfaces.
- Back-End Developer: Develops server-side logic and databases.
- Full-Stack Developer: Handles both front-end and back-end development.
- Web Application Developer: Creates dynamic web applications.
- JavaScript Developer: Specializes in JavaScript programming for web projects.
- Database Developer / Administrator: Manages and optimizes databases.
- Freelance Web Developer: Offers independent web development services.

References

[1].Kremer, M. (2022). *Introduction to Full-Stack Web Development: Node.js and the MEAN Stack* (Course Reader, 1st chapter). UC Berkeley Extension. https://ucbxwebsite.z13.web.core.windows.net/docs/FullStackWebDev_CourseReader_FirstChapter.pdf
[2]. Deitel, H. & Deitel, P. (Eds.). (2012). *The Complete Reference HTML & CSS* (5th ed.). McGraw-Hill Education."
[3]. Duckett, J. (2011). *HTML & CSS: Design and Build Websites*. Wiley.
[4]. Chacon, S., & Straub, B. (n.d.). *Pro Git* (PDF). Retrieved from https://kolegite.com/EE_library/books_and_lectures/Программирование/progit.pdf
[5]. *Full Stack Development (R20A0516): Lecture Notes, B.Tech III Year – II Sem (R20)*. Maisammaguda, Telangana, India. Retrieved from https://mrcet.com/downloads/digital_notes/CSE/III%20Year/AIML/Full%20Stack%20Development-Digital%20Notes.pdf

[6]. Svekis, L., van Putten, M., & Percival, R. (2021). *JavaScript from Beginner to Professional*. Packt Publishing. Retrieved from https://dn721809.ca.archive.org/0/items/Vismay/1425_JavaScript-from-Beginner-to-Professional.pdf
[7]. Rascia, T. *Understanding the DOM — Document Object Model*. DigitalOcean. Retrieved from https://assets.digitalocean.com/books/understanding-the-dom.pdf
[8]. Subramanian, V. (2019). *Pro MERN Stack: Full Stack Web App Development with Mongo, Express, React, and Node* (2nd ed.). Apress. ISBN-13: 978-1-4842-4391-6
[9]. Holla, P. (2016). *Express.js Guide: The comprehensive guide to Express.js*. Retrieved September 25, 2025, from https://pepa.holla.cz/wp-content/uploads/2016/08/Express.js-Guide.pdf
[10]. GeeksforGeeks. (2025, September 19). *Node.js CRUD Operations Using Mongoose and MongoDB Atlas*. Retrieved from https://www.geeksforgeeks.org/node-js/node-js-crud-operations-using-mongoose-and-mongodb-atlas/

KP-RETP Component 2: Classroom SECAP Evaluation Checklist

Purpose:

To ensure that classroom-based skills and entrepreneurship trainings under KP-RETP are conducted in an environmentally safe, socially inclusive, and climate-resilient manner, in line with the Social, Environmental, and Climate Assessment Procedures (SECAP).

Evaluator:________________________

Training Centre / Location:________________________

Trainer:________________________

Date: ________________________

Category	Evaluation Points	Status		Remarks /Recommendation
		Yes	NO	
Social Safeguards	Is the training inclusive (equal access for women, youth, and vulnerable groups)?			
	Does the classroom environment ensure safety and dignity for all participants (no harassment, discrimination, or child Labor)?			
	Are Gender considerations integrated into examples, discussions, and materials?			

	Is the Grievance Redress Mechanism (GRM) process, along with the relevant contact number, clearly displayed in the classroom			
	Are the Facilities and activities being accessible and inclusive for specially-abled (persons with disabilities)			
Environmental Safeguards	Is the classroom clean, ventilated, and free from pollution or hazardous materials?			
	Is there proper waste management (bins, no littering)			

	Are materials used in practical sessions environmentally safe (non-toxic paints, safe disposal of wastes)?			
	Are lights, fans, and equipment turned off when not in use (energy conservation)?			
Climate Resilience	Are trainees oriented on how their skills link with climate-friendly practices (e.g., renewable energy, efficient production, recycling)?			
	Are trainers integrating climate-smart			

	examples in teaching content?			
	Are basic health and safety measures available (first aid kit, safe exits, fire safety)?			
	Is the trainer using protective gear or demonstrating safe tool use (where relevant)?			
Institutional Aspects	Is SECAP awareness shared with trainees (via short briefing, posters, or examples)?			
	Are trainees encouraged to report unsafe, unfair, or environmentally harmful practices?			

Overall Compliance	Overall SECAP compliance observed	☐ High ☐ Medium ☐ Low	

Overall remarks/ recommendations			
Name	**Designation**	**Signature**	**Date**